Neural Networks that Learn Symbolic and Structured Representation of Information

Itsuki Noda

1994

Kyoto University

Abstract

In the research of artificial intelligence, symbol processing has been providing powerful tools to represent and process complicated and varied information. However, it has disadvantages in '*analogy*', '*uncertainty*', and '*learning/adapting*'. Neural networks have been expected to conquer these disadvantages of symbol processing. In order to provide flexible and robust problem solving methods, many researchers have been trying to construct hybrid or integrated systems of symbol and neural processing.

However, there remains a problem in such a hybrid way, which comes from difference of characteristics of data that neural and symbol processings deal with. Especially, following two essential characteristics of data representation in symbol processing are important.

- *Symbols*: each of which indicates discrete and independent information.
- *Data structures*: by which complicated information is arranged flexibly.

Because neural networks originally have not mechanisms to deal with these characteristics, it is difficult to transfer information flexibly between neural and symbol processing modules in a hybrid system.

In this thesis, I describe learning methods of neural networks that provide a way to deal with *symbols* and *data structures*.

First, the symbolization of patterns in neural networks is investigated. In order to analyze patterns as symbols, it is useful that pattern representation is simple and has no redundant part. However, networks get redundant representation through its learning, so that such analysis becomes difficult.

In chapter 2, a new learning method, '*overload learning*', to solve the problem of how to eliminate such redundant representation is described. In this method, a network is trained to learn an additional task together with an original one. Since a redundant part of pattern representation is used for the additional task, only minimum representation

becomes to be used for the original task. Various experiments show that the proposed method makes the symbolization of patterns easy.

Second, pattern representation of data structures is investigated. The variety of the size of data is one of the major causes of the difficulty of processing structured data by neural networks. Although the size of structured data generally varies, neural networks usually process fixed-sized patterns. Temporal-sequence processing is a technique to process such a variable-sized data by using processors that can process fixed-sized data. Yet, in such a technique the processors need to learn to process sequences that have *long distance dependencies* (LDD).

In chapter 3, learning methods for simple recurrent networks to solve the problem of how to find LDDs are described. In order to find LDDs, a simple recurrent network needs to retain information about input histories in patterns. I formalize two measures of how much information is retained in patterns. In the first formalization, a measure of the information is defined by distances between patterns. Using this measure, the '*distance-keeping*' method is proposed. In the second formalization, a measure of loss of the information is defined in the manner of Shannon's information theory. Based on this measure, '*information-loss minimization*' method is proposed. Experiments show that both methods increase the ability to deal with LDDs as compared with a conventional back-propagation learning.

Temporal sequence processing provides another point of view for representing structured data. Simple recurrent networks have a similar structure to finite state transducers. On the other hand, in the automata theory, state-transitions of a transducer represent a structure of sequences which the transducer processes. In the same way, simple recurrent networks have the ability to represent a structure of the sequences. However, they can not acquire suitable state-transitions by conventional learning methods.

In chapter 4, a model, called the '*SGH model*', and its learning method are proposed. They construct a simple recurrent network that has suitable state-transitions for a given task. They are derived from a procedure to construct a finite state transducer using the state-minimization technique. Experiments show that the SGH model can acquire suitable state-transitions for given tasks.

In chapter 5, I discuss about proposed models and methods from various points of view. First, the ability of simple recurrent networks and that of finite state transducers are compared. Because of a topology of patterns, the flexibility of state-transitions of simple recurrent networks is limited as compared with finite state transducers. On the

other hand, the topology increases the generalization ability of learning state-transitions. I show this advantage through an experiment to deal with *sub-grammars*. Second, the formalization of a semantic network that is suitable for processing by a simple recurrent network is discussed. In this formalization, a semantic network is treated as a chart of state-transitions of a simple recurrent network. Finally, biological plausibility of proposed models is discussed. While artificial neural networks are originally derived from biological nervous systems in brains, many of them are not plausible as nervous systems. Proposed methods and models are simple enough and relatively plausible as biological models from various points of view.

Acknowledgments

I would like to acknowledge my sincere appreciation to Professor Makoto Nagao of Kyoto University for his supervision and continuous encouragement.

I would like to express my greatest gratitude to Professor Satoshi Sato of Advanced Institute of Science and Technology, Hokuriku for his constructive discussions and continuous support.

I would also like to thank Professor Jun-ichi Tsujii of the University of Manchester, Professor Jun-ichi Nakamura of Kyushu Institute of Technology, Professor Yuji Matsumoto of Advanced Institute of Science and Technology, Nara and Professor Yuichi Nakamura of Tsukuba University for their stimulating discussions and valuable advice when they are at Kyoto University.

I am grateful to Motoi Suwa, Director of Information Science Division of Electrotechnical Laboratory and Kazuhisa Niki, Chief of Cognitive Science Section for their support. I am also grateful to my colleagues in Electrotechnical Laboratory, especially Dr. Hideyuki Nakashima and Dr. Hitoshi Matsubara.

I am grateful to all previous and current members of Professor Nagao's laboratory, especially Dr. Yasuharu Den of ATR Interpreting Telecommunications Research Laboratories.

I would like to thank Dr. Alan W. Black of ATR Interpreting Telecommunications Research Laboratories for his helpful comments on a draft of this thesis.

Contents

Chapter 1

Introduction

1.1 Symbol Processing and Neural Processing

Researches of artificial intelligence (AI) aim to understand human information processing and to realize human intelligence on computers. Human information processing has the following features.

- Complexity
  A human can process complicated information. Such processing is also complicated.
- Variety
  A human can process various information.
- Analogy
  A human can process unknown information by using analogy.
- Uncertainty
  A human can process noisy and ambiguous information.
- Adaptability
  A human can adapt his processing to various environments and acquire concepts.

In works in AI on symbol processing, the complexity has been focused first. Symbols and data structures are powerful tools to process such complex information. *Symbols* provide high operationality of information because they are discrete and independent from each other. *Data structures* provide a rich framework to represent complicated information. Complicated information is arranged by using data structures. Such information is processed with the help of the operationality of symbols.

Data structures are also useful to represent various information. Using data structures, various information is composed of pieces of information flexibly. Symbols and data structures have brought many successes to AI research.

On the other hand, other features are weak points of symbol processing. In order to process information by means of analogy, *similarities* of information must be defined. Since any similarities are originally not defined between data in symbol processing, we need additional mechanisms to deal with similarities of information. Memory based reasoning is one of such mechanisms. In memory based reasoning, however, there also remain problems: how to define similarity between symbols and how to deal with various data structures.

Symbol processing also requires mechanisms to deal with uncertainty. Probabilities and the fuzzy theory are widely used as such mechanisms. It is easy to implement them into symbol processing and also easy to understand them intuitively. However, representation of uncertainty is limited by symbols and data structures, because such mechanisms are implemented as a property of symbols and data structures. In other words, we can only represent uncertainty of information that is represented explicitly by symbols and data structures. Therefore we must carefully define what kind of information, symbols and data structures represent. However, there has not been a general method that tells us what should be represented by them. In addition, we need another mechanism to interpret raw data from environments, such as sensor inputs, which are main sources of uncertainty.

Machine learning has been heavily investigated in AI in order to adapt processing to various environments. However, there remain severe problems. One problem is that most machine learning systems are domain-dependent and are not robust to noise. Another problem is that the ability of machine learning is strongly dependent on the representation of information. Therefore we must carefully define such representation. However, general methods how to define suitable representation have not been proposed.

In order to conquer these weak points of symbol processing, neural networks have been given attention. Neural networks process patterns, which have a topology. Using this topology, similarities of information are defined. Thus neural networks can process information according to such similarities naturally. Pattern representation also provides a method to deal with uncertainty. For example, it is easy to interpret an activation of each element of a pattern as a probability or a possibility of certain information. Moreover, Neural networks have powerful learning methods to adapt themselves to various

environments. Most of these methods are general-purpose and robust to noise. In addition to it, networks can acquire suitable internal representation for given tasks through learning.

However, neural processing is bad at dealing with complexity and variety of human information processing. Since pattern representation is *flat*, it is difficult to represent structured data explicitly like symbol processing. Moreover, a piece of information is distributed in a pattern. Therefore it is not easy to compose various information from pieces of information.

In order to complement weak points of symbol and neural processing, hybrid or integrated systems of these two kinds of processing are focused. In such systems, symbol processing provides methods to deal with complexity and variety, and neural processing provides methods to deal with analogy, uncertainty and adaptability. Such systems are expected to provide flexible problem solving methods like human information processing.

However, there is a crucial problem of how to transfer information between symbol and neural processing. The type of representation of information used in both processings are quite different. Therefore it becomes a bottleneck to transfer information between symbol and neural processing modules. The main cause of this problem is that it is difficult to represent *symbols* and *data structures* in a symbol processing module by patterns in a neural processing module. In order to provide a method of tight communication between those modules, I pay attention to the following two points:

- How to analyze patterns as *symbols*.
- How to deal with *data structures* by neural networks.

1.2 Symbols

Consider a hybrid system of neural and symbol processing. When the neural module transfers information to the symbol module, the system needs to analyze pattern representation in the neural module as symbols. Generally patterns do not represent symbols explicitly, so that the analysis becomes a major problem of hybrid systems.

One way to analyze patterns as symbols is *clustering*. Patterns in a neural module are classified into clusters by various clustering techniques. Then, each cluster is interpreted as a symbol in the symbol module. Many researchers have been using these techniques in order to analyze what neural networks learn [Elm88, Pol90, SSCM89]. In such cases,

clusters of patterns are required to be reduced and separated from each other clearly. In fact, however, networks acquire redundant pattern representation, so that patterns do not form reduced clusters.

Another way to analyze patterns is to deal with each unit in a neural network as a certain primitive symbol and the activation of the unit as a probability of the symbol. In this technique, each unit is required to be suitable as a primitive element that is independent from each other. In fact, however, it is not assured that all units become independent after learning of a neural network. Therefore some units remain redundant. In this case, it is difficult to find which units are primitive.

A fundamental issue underlying these difficulties is:

How to eliminate redundant representation which networks learn.

This is also a general problem of neural processing because this issue concerns the problem of generalization ability of network learning.

1.3 Data Structures

Consider a case in which a neural module receives data from a symbol module in a hybrid system. In this case we need to represent structured data by patterns that can be processed by neural networks. A conventional technique to represent data structures in neural networks is to construct neural networks that have the same structures as the data. In this technique, however, the network can not manipulate or learn structures.

One of the major problems of representing structured data by patterns is the variety in the size of representation. The size of patterns which neural networks process is fixed, while the size of structured data generally varies. Temporal sequence processing is a technique to deal with variable-sized data by a processor which processes fixed-sized data. Therefore we can solve the problem by processing structured data as temporal sequences by using recurrent neural networks.

Yet, the 'temporal processing' technique gives rise to another problem, that is the '*long distance dependency* (LDD)' problem: In temporal sequence processing, inputs might be given long before processors require information about the inputs. In this case processors must keep information about inputs until it becomes needless. For example, when a processor checks an agreement of a subject noun and its verb in a sentence such as

"*The dog* which chased cats *is* mine",
"*Dogs* which chased cats *are* mine",

the processor must keep information about a subject noun during a relative clause. In order to solve this problem, a technique to find LDDs is required. In the context of neural networks, we must solve the problem of how to train recurrent networks to process temporal sequences that have LDDs.

Temporal processing provides another view of representing and processing structured data. In automata theory, state-transitions of a transducer that processes sequential data represent sequentized structures. On the other hand, it has been pointed out by many researchers that a recurrent network can be treated as a finite state transducer. Therefore we can consider a method in which a recurrent network can deal with data structures through state-transitions of the network. In order to complete this method, we need a technique to learn suitable state-transitions of recurrent networks.

1.4 Biological Plausibility

In the research of neural processing, biological plausibility is an important point of consideration.

The idea of neural processing comes from biological models of actual nervous systems. However, many of artificial neural network models are not biologically plausible. For example, the 'back-propagation through time' method [WZ89] requires to store all states of the networks during processing. Such mechanisms are not plausible as actual nervous systems [GA91].

In works described in this thesis, such plausibility is considered carefully, especially, with respect to the following points.

- Locality of calculations
  Locality of calculations is one of the important features of neural processing. It is said that there are no global supervisors that control calculations in biological nervous systems. Instead of them, calculations are realized as cooperations between neurons. Such cooperations are done in physically local area. Thus, in artificial models of neural networks, calculations should be done locally. For example, calculations of the back-propagation learning are local, so that it is plausible in this sense.

Time-locality is also important. No mechanisms to memorize activation patterns in neural networks over time have been found. Therefore all calculations should be done using current activation patterns. This becomes an important issue when a neural network processes temporal sequences. For example, the 'back-propagation through time' method is not time-local, because it requires all past activations of networks. Thus it is not plausible.

- Penalty functions
  Learning procedures such as the back-propagation method may also be implausible as an actual learning procedure in nervous systems. In order to make models independent from learning procedures, I focus penalty functions that are minimized by such learning procedures. In learning methods proposed in this thesis, only penalty functions are modified instead of learning procedures. This strategy makes it easy to implement these methods using other learning procedures. Note that such penalty functions should be calculated locally.

1.5 Outline of the Thesis

In this thesis, I describe learning methods for neural networks to solve the following problems described above:

1. How to eliminate redundant representation.
2. How to find LDDs in temporal sequences.
3. How to learn suitable state-transitions.

In chapter 2, I describe a method to solve the first problem. First I consider that redundant representation is caused by too many hidden units for a given task. In other words, the issue is how to balance the number of units and the complexity of a task. In order to balance them, I propose a method, called the 'overload learning' method. In this method, a network is trained to learn an additional task together with an original task. Because learning of the additional task requires to use hidden units, the number of units that are used for the original task decreases. If we can control learning of both tasks, the network will use the suitable number of units for the original task. I also explain results of various experiments and show effects of this method.

In chapter 3, I describe two methods to solve the second problem. I consider that a recurrent network needs to keep information about input histories on hidden layers as long as possible in order to find LDDs. Then I formalize measures of information represented by patterns of hidden layers in two ways. Based on these measures, two methods to decrease loss of the information about input histories are proposed. In one way, information is measured by the distance between patterns. I analyze the relation between the change of the distance and a distribution of weight values of links. Based on this relation, a method, called the 'distance-keeping' method is derived. In another way, loss of information is defined in the manner of Shannon's information theory. I show the relation between the loss of information and learning of identity functions by three-layered networks. Based on this relation, a method, called the 'information-loss minimization' method is derived.

In chapter 4, I describe a method to solve the third problem. Initially a procedure to construct a finite state transducer from examples of input-output sequences is composed using the state-minimization technique. Then each step of the procedure is reconstructed as learning of a neural network. Finally those networks and their learning methods are combined into a model, called the 'SGH model.' The ability of the SGH model is demonstrated through experiments of learning various state-transitions.

In chapter 5, I discuss about proposed models and methods from various points of view. First the comparison of simple recurrent networks and finite state transducers is discussed. Although a simple recurrent network can be treated as a finite state transducer as discussed in chapter 4, abilities of them are slightly different. In this discussion, I focus on the generalization ability and flexibility of state-transitions. Second, I propose a prototype of the formalization of semantic networks that are suitable to process by simple recurrent networks. Semantic networks are a framework to represent various information used in symbol processing. This formalization will provide a way to combine neural and symbol processing tightly. Finally, the biological plausibility of proposed models is discussed. While neural networks are originally derived from biological nervous systems in brains, many artificial neural networks are not so plausible as actual brain models. I examine proposed methods and models from various points of view.

Chapter 6 outlines the conclusions of this thesis.

Chapter 2

Symbolization by Overload Learning

2.1 Introduction

In order to integrate symbol processing and pattern processing in neural networks, we need to analyze pattern representation in neural networks from the view point of symbols. One way to analyze patterns is to deal with each dimension of the pattern space as an independent micro-feature. In this case each dimension is desired to be independent from each other. Thus it is necessary that there are no redundant dimensions in the pattern space. In other words, it is necessary to reduce the number of dimensions of patterns, especially active dimensions of patterns, so that the number of active dimensions is suitable for representing information for a given task. Generally, however, it is difficult to determine how many dimensions are required for given tasks.

Another way to analyze patterns is to classify patterns into clusters by clustering methods. After clustering, each cluster is interpreted as an individual symbol [OGM92, Elm91, CSSL89]. In this case, clusters of patterns are required to be separated clearly. Especially, clustering will become easy when each cluster of patterns is converged into small compact area. However, clusters tend to spread if there is room in the pattern space. Cluster spreading upsets suitable clustering of patterns.

Furthermore, in the general purpose of neural networks, it is important to find the minimal number of units or dimensions for given tasks. The reason is that minimization of the number of dimensions increases the generalization ability of learning. Unfortunately, it usually isn't obvious what size is best for a given task.

These problems come from one issue: how to balance the capacity of networks and the complexity of tasks. In order to solve this issue, a network should contain a suitable

number of hidden units for a given task. On the other hand, a network is trained with too many hidden units because of the guarantee of successes of learning. A conventional way to solve this issue is to adjust the capacity of a network such as the number of units to the complexity of a task[WK90, Hag91] by using certain criterion or pruning methods. In this chapter, I propose another way in which the complexity of a task is adjusted to the capacity of a network.

2.2 Overload Learning

2.2.1 Activeness and Redundancy

Suppose that we train a network to learn a given task. We say a unit is '*active*' for the task when the activation of the unit changes for various inputs of the task, and '*inactive*' for the task when the activation does not change. Moreover, we say a unit is '*redundant*' for the task when the network can achieve the task without that unit.

We use these three words, 'active', 'inactive' and 'redundant', not only for units but also for dimensions of a vector space of hidden patterns. For example, we say a dimension is active for the task when a position of a hidden pattern changes along the dimension in the pattern vector space for various inputs.

When we train a network to learn a given task, in order to guarantee the success of the learning process, we usually use a network that has more hidden units than the task will require. In this case there remain some redundant units or dimensions in the hidden layer. In those redundant units/dimensions, inactive units/dimensions are not so important, because they do not affect the action of the network or the ability of generalization. On the other hand, redundant and active units/dimensions are important. They should be eliminated because of the following reasons:

- Redundant and active units/dimensions decrease the generalization ability of learning. Redundant and active units/dimensions increase degrees of freedom of the representation. Generally, more degrees of freedom of the representation bring less generalization. Therefore redundant and active units/dimensions upset the generalization.

- They upset analysis of patterns of a hidden layer. Usually, active units/dimensions are treated as primitives each of which indicates independent element of information

Figure 2.1: Three-Layered Networks

from each other. But redundant units/dimensions indicate duplicate information. It is difficult to eliminate such duplicate information at a stage of the analysis.

2.2.2 How to Eliminate Active and Redundant Dimensions

Consider a case in which a three-layered network shown in Fig. 2.1 is trained to learn a given task (called an '*original task*'). As mentioned in the previous section, some dimensions in the hidden layer become redundant and active for the task. Next, consider a case in which new input and output layers are added to the network as shown in Fig. 2.2. The network is trained to learn another task (called an '*additional task*') using these additional input and output layers. This learning is done at the same time of the learning of the original task. In this case, redundant dimensions for the original task will become used for achieving the additional task. When the additional task is independent from the original task, redundant and active dimensions for the original task become inactive for the original task. Therefore, we can eliminate active and redundant dimensions by training the network to learn an additional task simultaneously by adding additional input and output layers. I call such a method '*overload learning* (OLL).'

In order to eliminate active and redundant dimensions effectively, the additional task should satisfy the following conditions:

1. The additional task is independent from the original task.

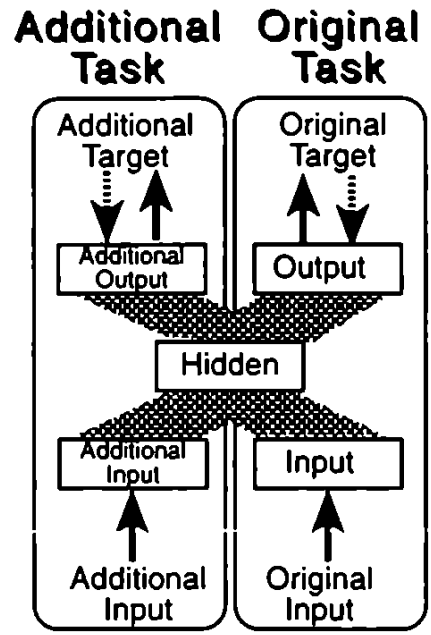


Figure 2.2: Three-Layer Networks with Additional Input/Output Layers

2. The additional task requires the larger number of hidden units than the given network.

3. A neutral input of the additional task is easy to prepare. After learning, the additional task is not necessary and upsets the analysis of patterns for the original task. Therefore we need a way to avoid effects of the additional task.

Moreover, we must control the learning of these two tasks in order to give priority to the original task over the additional task.

2.2.3 Overload Learning with Random Task

An identity mapping of random patterns (called a '*random task*') was chosen as an additional task. In this task, randomly generated patterns are put into the additional-input layer. The additional output layer is required to output the same pattern as inputs. This task can satisfy the three conditions for the additional task as follows. This task is obviously independent from any other task, so it can satisfy the first condition. The random task requires the same number of hidden units as the size of input patterns. Therefore the

second condition is satisfied when the size of the additional-input layer is larger than the size of the hidden layer. Moreover, the third condition is satisfied by using the average of all patterns as a neutral input.

For OLL with the random task, a network shown in Fig. 2.3 is used. This network is called an OLL network. In this network, *input* and *output* layers are used for the original task, and *random-input* and *random-output* layers are for the random task. The size of the random-output layer is the same as the size of the random-input layer, and larger than the size of the hidden layer. In experiments described in the next section, these sizes are twice of one of the hidden layer.

In the learning phase, an input pattern of the original task is set into the input layer and a required output pattern is given to the output layer as a target. At the same time, a randomly generated pattern is set into the random-input layer and the same pattern is given to the random-output layer as a target. After learning, the random-input layer is fixed to the average pattern of random patterns. The network is trained by back-propagation to minimize the following penalty function.

$$E = < |o_{\text{output}} - t_{\text{output}}|^2 > + \alpha < |o_{\text{rand-output}} - o_{\text{rand-input}}|^2 > \qquad (2.1)$$

where o_{output}, $o_{\text{rand-output}}$ and $o_{\text{rand-input}}$ are output patterns of output, random-output, random-input layers respectively; t_{output} is the target pattern of the original task. $< x >$ means an average value of x; $|x|$ is the norm of vector x; α is a positive coefficient. In (2.1), the first term of the right side is the penalty for the original task and the second term is the penalty for the additional task. In order to give priority to the original task over the random one, α is set relatively smaller than 1. In this case, the penalty for the original task shares a major part of E, so that the network learns the original task primarily and the random task secondarily. In the following experiments, α is set $0.3 \sim 0.5$.

2.3 Experiments and Discussion

2.3.1 Finding Minimum Dimension

In order to determine the kind of effects of OLL over forming pattern representation on the hidden layer, the following experiment was carried out: Initially, a target network, whose input, hidden and output layers consist of n, m and l units respectively, is created. Using this target network, 1000 input-output pairs are generated. Then an OLL network,

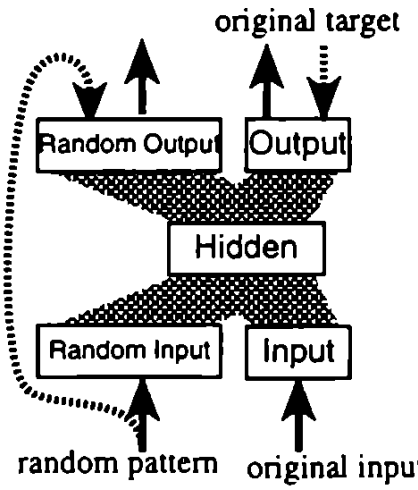


Figure 2.3: Structure of OLL Networks

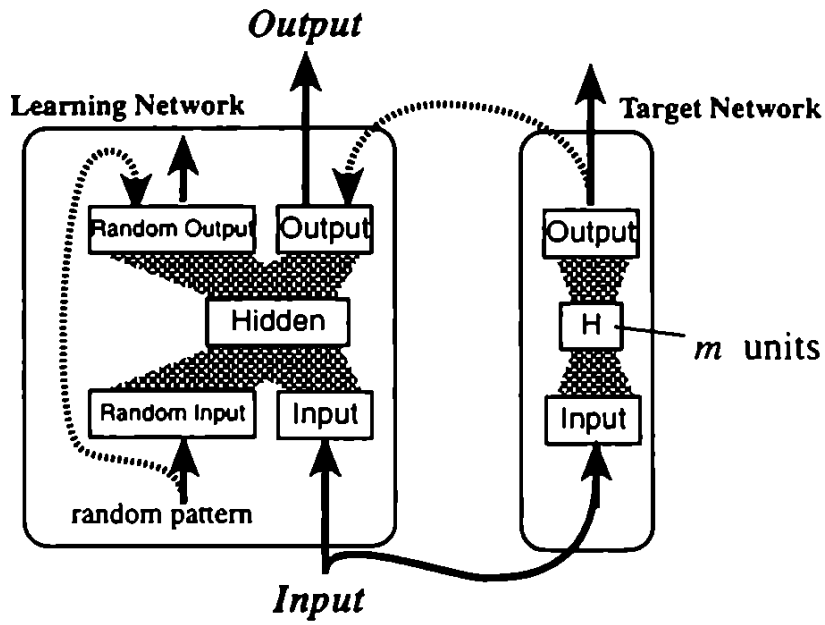


Figure 2.4: Setup of Experiment of Finding Minimum Dimension

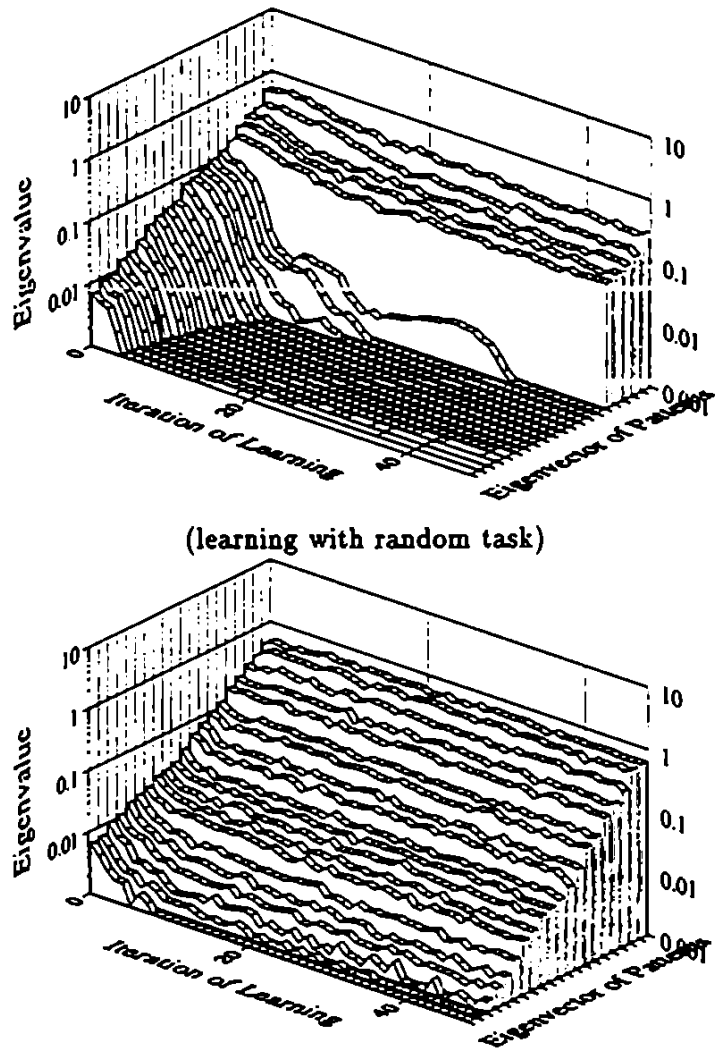
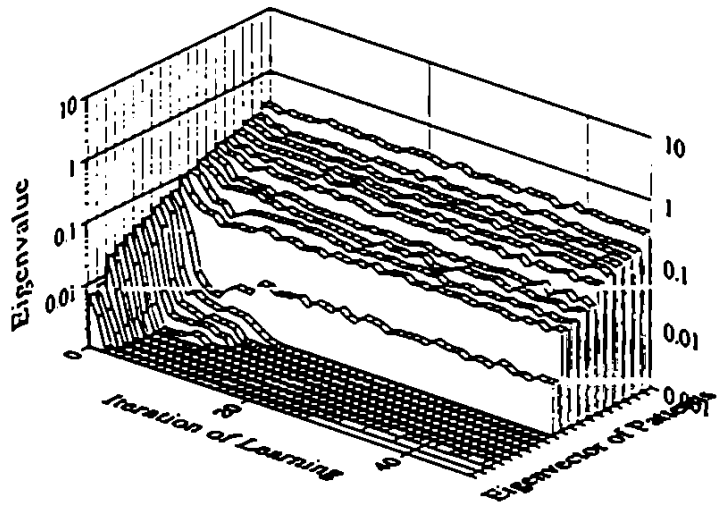
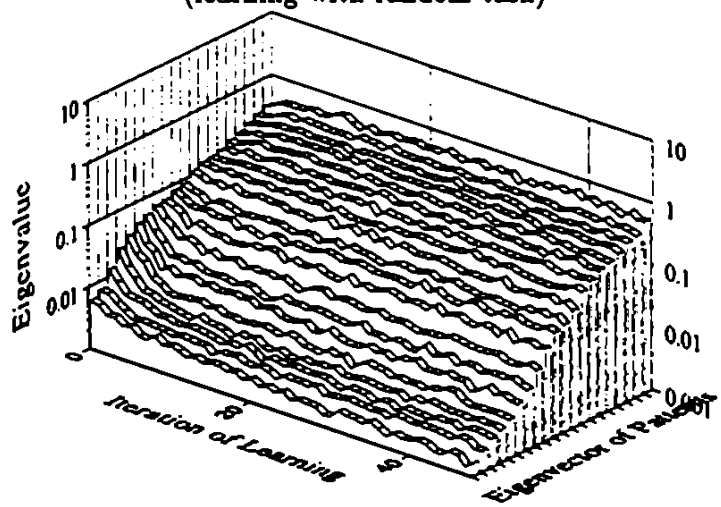


Figure 2.5: Eigenvalues of Hidden Patterns. (target: 5 hidden units) Eigenvalues are shown in log scale.



(learning with random task)



(learning without random task)

Figure 2.6: Eigenvalues of Hidden Patterns. (target: 10 hidden units) Eigenvalues are shown in log scale.

whose input, hidden, output and random-input (also random-output) layers consist of n, k, l and $2k$ units respectively, is trained by these examples with a random task. Fig. 2.4 shows the set up of this experiment. After training, patterns on the hidden layer for various inputs are analyzed by principle component analysis (PCA). When a certain eigenvector of PCA has a significant eigenvalue, the dimension along the vector is *active* for the task.

Fig. 2.5 shows a result of the analysis where $n = 20$, $m = 5$, $l = 20$ and $k = 30$. This graph shows changes of the 1st ~20th eigenvalues of hidden patterns through learning. In the case of learning with the random task (the upper graph), that is OLL, only up to the 5th eigenvalues remain significant, and the rest become very small as the learning proceeds. This means that only 5 dimensions are *active* for the task. On the other hand, without the random task (the lower graph), that is a conventional learning, all eigenvalues remain significant. Fig. 2.6 shows another result where $m = 10$. In this case, 11 eigenvalues remain significant by OLL.

From these results, we can say that OLL reduces the number of *active* dimensions of hidden patterns into the suitable number of dimensions for the original task.

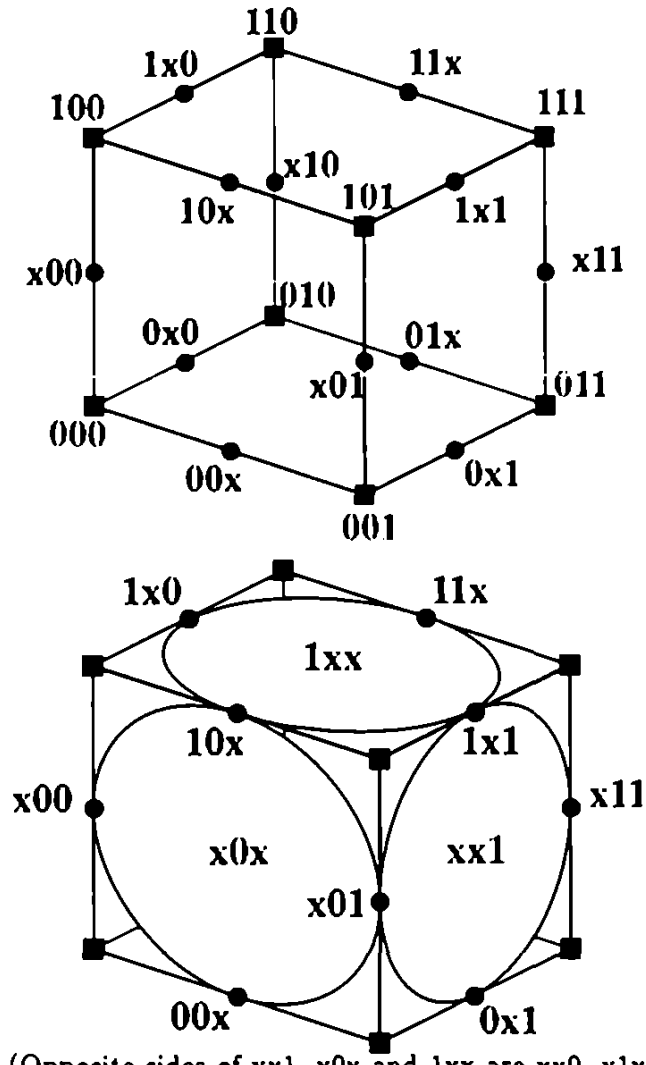
2.3.2 Finding Primitive Dimension

In order to show that a network can find primitives of information about a given task by OLL, the following experiment was carried out.

Consider that each apex, edge and plane of a cube is labeled as shown in Fig. 2.7, and make a mapping from an edge to two planes and two apexes that connect to the edge. For example, an edge '10x' is mapped to planes '1xx, x0x' and apexes '100, 101.' This mapping is represented by patterns using *localist representation* as shown in Fig. 2.8. Then a network shown in Fig. 2.9 is trained to learn this mapping by OLL. In this network, 12 units is used for the edge layer, 12 units for the hidden layer, 6 units for the plane layer, 8 units for the apex layer and 24 units for the random-input layer and random-output layer. In order to test the generalization ability, underlined patterns in Fig. 2.8 are not given to the network as targets during learning.

Fig. 2.10 shows outputs of the apex and plane layers after learning. Fig. 2.10-(a) is the case of OLL and (b) is the case of the conventional learning. In the case of the conventional learning, outputs which the network has not been taught are not correct, while all outputs are correct in the case of OLL. This result shows OLL increases the generalization ability for this task.

Why does this generalization arise? I analyzed hidden patterns for each edge label



(Opposite sides of xx1, x0x and 1xx are xx0, x1x and 0xx respectively.)

Figure 2.7: Labelling of Edges, Apexes and Planes of a Cube.

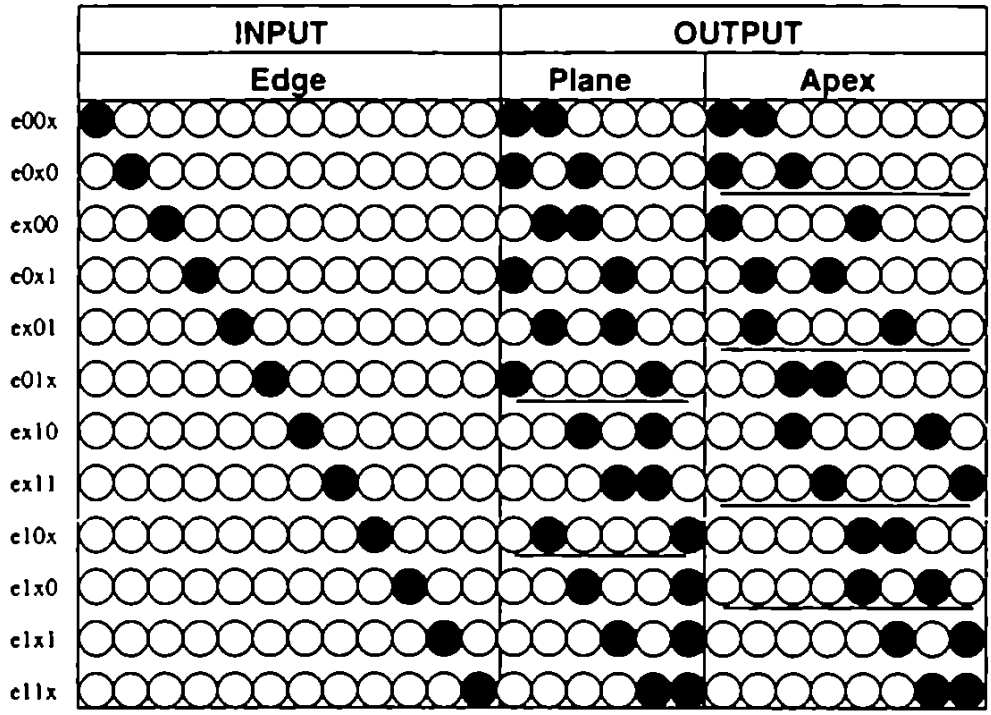


Figure 2.8: Mapping from A edge to Planes and Apexes (Cube Mapping)

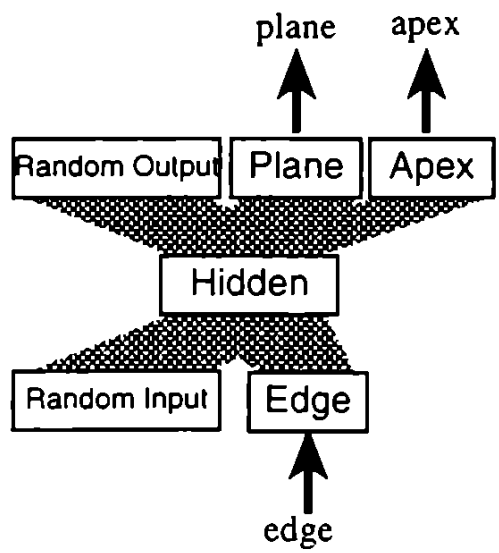


Figure 2.9: Network for Learning Cube Mapping

input by PCA. Then I found only 3 dimensions remain *active* in hidden pattern space. In the 3 dimensional space each pattern for each edge label is arranged like Fig. 2.11-(a). This arrangement is the same as of midpoints of edges of a cube (Fig. 2.12). On the other hand, the arrangement of patterns in the case of the conventional learning is like Fig. 2.11-(b). In this case, it is difficult to find correspondence of Fig. 2.11-(b) and Fig. 2.12.

From this result, we can say that the OLL network found three primitive dimensions of the task and represented information by patterns suitable for the task. Then the network became able to generalize the task and responded for unknown output correctly.

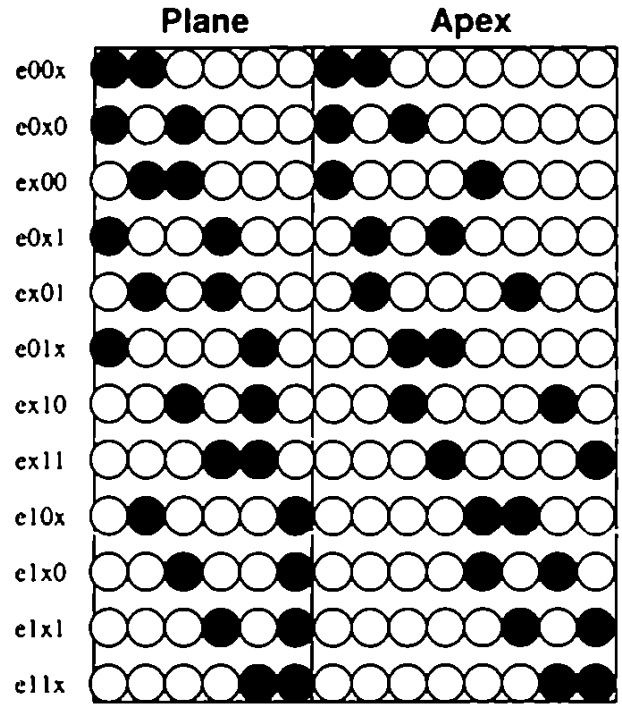
2.3.3 Converging of Pattern Clusters

One purpose of OLL is to make internal representation compact. In order to analyze hidden patterns as symbols by clustering techniques, it is important that these patterns form clearly-separated clusters. For example, many researchers have been trying to analyze pattern transitions of simple recurrent networks[OGM92, Elm91]. It is, however, difficult to find a clear structure of transition because patterns do not form compact clusters in pattern spaces.

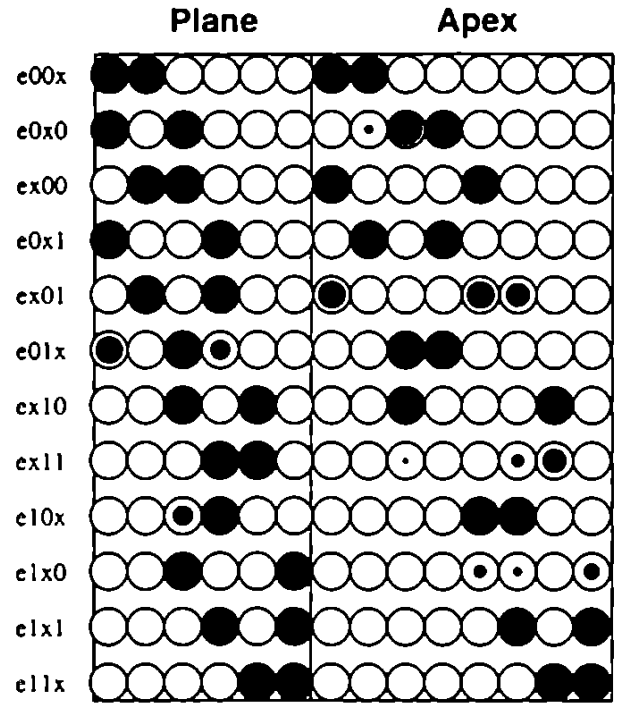
On the other hand, OLL is expected to have an effect on convergence of these clusters. This effect is led as follows. When a cluster of patterns for the original task gets expanded, it will behave as noise to the random task. Therefore, the network learns to accommodate the random task by converging the cluster into compact areas.

In order to demonstrate this effect, the following experiment was carried out: A simple recurrent network with a random task (Fig. 2.13) is trained to predict the order of characters in sequences [Elm88]. At each point in time, a character in a sequence generated by the Reber grammar (Fig. 2.14 [CSSL89]) is presented to the network. The network's target output is simply the next character in the sequence. For example, when the Reber grammar generates a sequence 'TSXXVPS', the network receives an input sequence 'TSXXVPS' and a target sequence 'SXXVPSσ'. (σ indicates the end of a sequence.) In this learning, the network is trained to achieve the task with a random task in the same manner as OLL. The experiment was carried out using the network whose input, hidden and output layers consisted of 10, 30 and 10 units respectively. (Each of random-input and random output layers consisted of 60 units.)

After learning, hidden patterns during processing sequences of the Reber grammar were analyzed by PCA. The result of the analysis is shown in Fig. 2.15. In this figure,

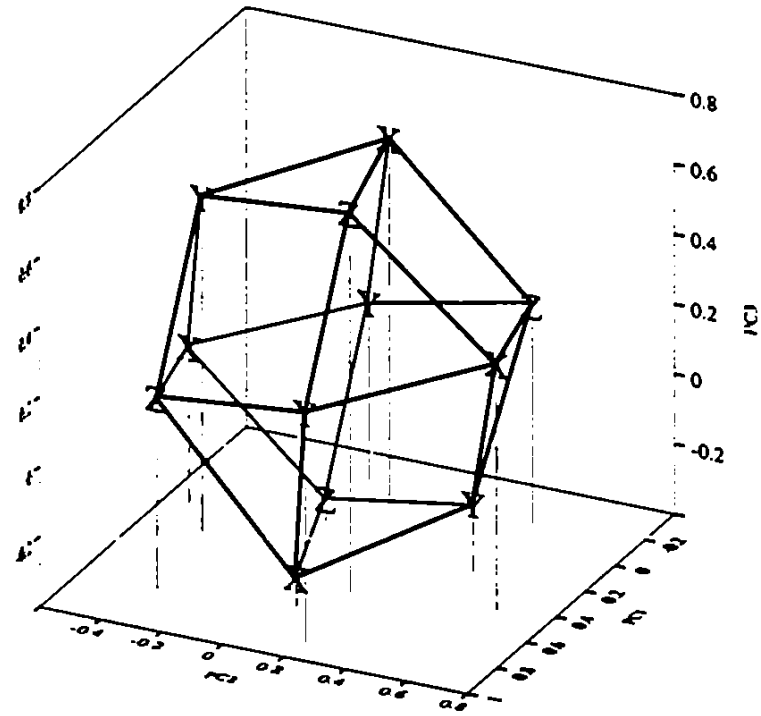


(a) With Random Task

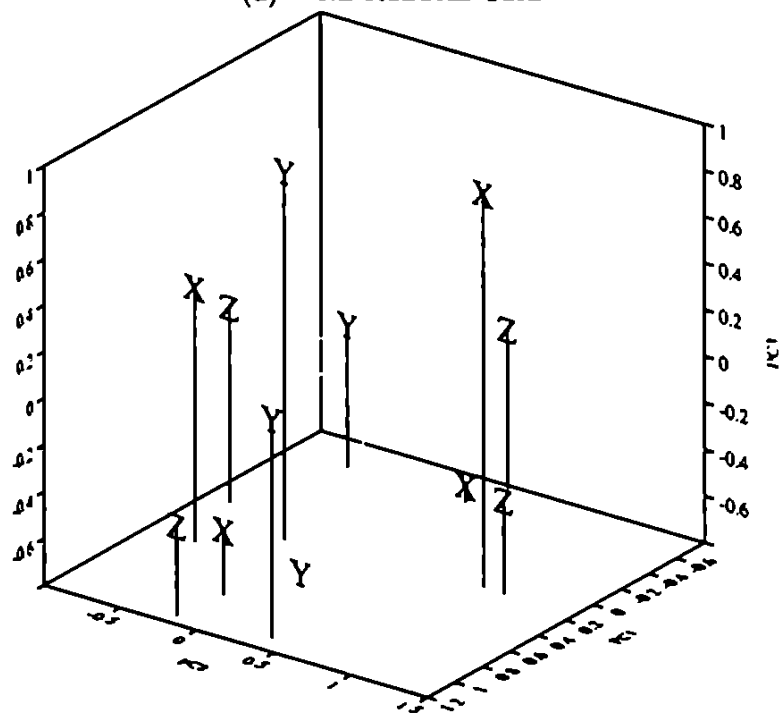


(b) Without Random Task

Figure 2.10: Results of Learning of Mapping



(a) With Random Task



(b) Without Random Task

Figure 2.11: Locations of Hidden Patterns of Each Edge Input.

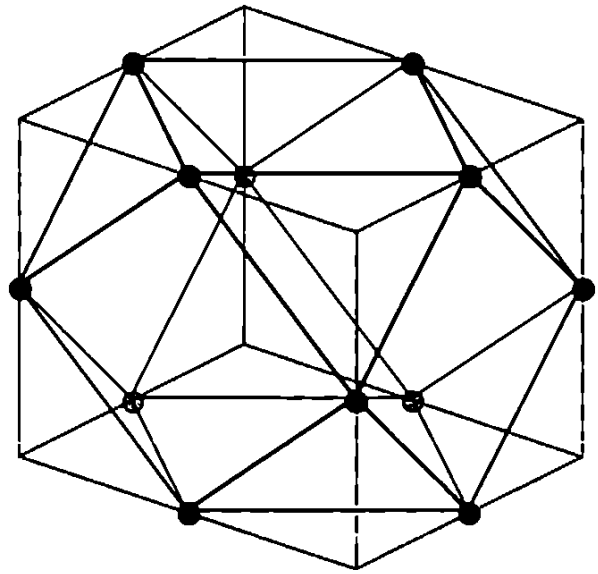


Figure 2.12: Locations of Mid Points of Edges of a Cube.

each letter in the graph means the position of a pattern corresponding to the state in Fig. 2.14 whose number is the same as the letter. In case of the conventional learning (the lower graph), patterns form some clusters. But these cluster are spread and mixed complexly. On the other hand, OLL converges clusters of patterns into very compact areas in the manner described above. As a result, patterns in the same cluster are identical. Therefore, it seems that there are very few points in the space in the upper graph (that is the case of OLL) although the same number of points are plotted.

2.3.4 Generalization by Clustering

In the field of machine learning, finding classes of data is an important topic. In this case, a '*class*' means that all data in the class share some properties. These classes are used for generalization in the following manner: When a datum has one of shared properties of a class, the datum is assumed to belong to the class. Then the datum is expected to have other shared properties of the class.

OLL is expected to have a similar generalization ability. OLL converges clusters of hidden patterns into very compact areas. Such compact clusters can be interpreted as classes of input patterns, because a network outputs similar patterns for inputs when

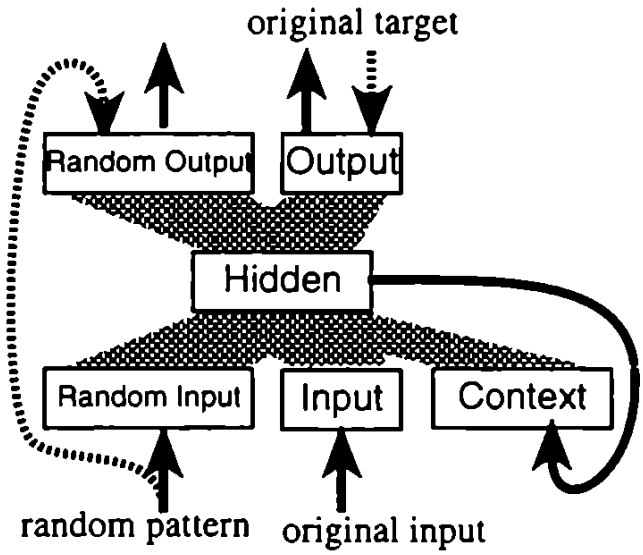


Figure 2.13: Simple Recurrent Network with Random Task

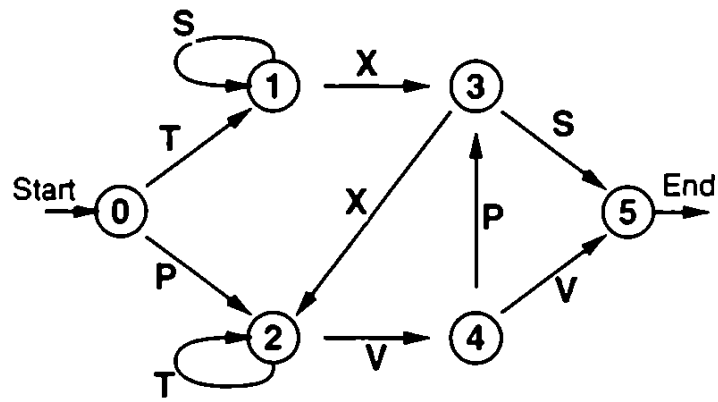
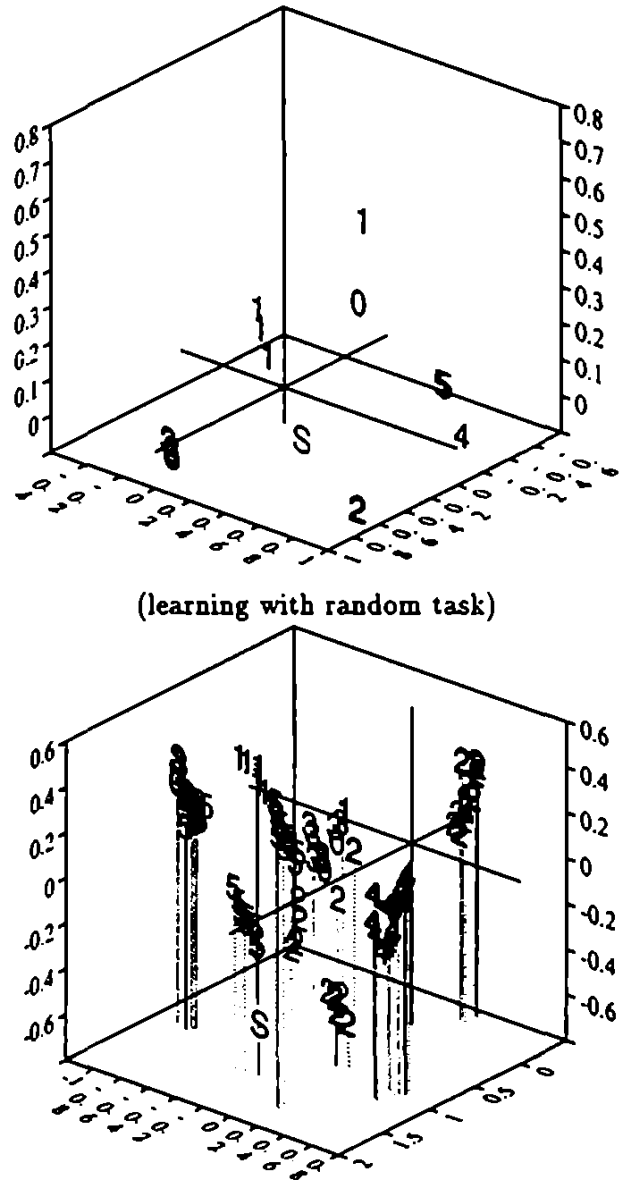


Figure 2.14: Reber Grammar



(learning with random task)

(learning without random task)

Figure 2.15: Distribution of State Patterns of Reber Grammar. Each graph shows the position of patterns in the 1st, 2nd and 5th principle components space.

hidden patterns for the inputs belong to the same cluster. Therefore when a network can acquire a suitable compact cluster of hidden patterns from incomplete information by OLL, we can say OLL has an effect of generalization by finding classes.

In order to test this effects, the following experiment was carried out: Initially, 16 patterns ($p_{Xi}|X \in \{A,B,C,D\}, i \in \{1,2,3,4\}$, size of pattern is 20) are generated randomly and two property values (q_{Xi}, r_{Xi}) are defined for each pattern p_{Xi}, where $q_{Xi} = q_{Xj}$, $r_{Xi} = r_{Xj}$ for any i,j. In other words, there are four classes of patterns (that are class 'A', 'B', 'C' and 'D') each of which has the same pair of property values. Then an OLL network, which consists of two output layers for property values q_{Xi}, r_{Xi} and 10 hidden units, is trained to output q_{Xi}, r_{Xi} when it receives p_{Xi}. But the network is not taught about q_{X4}, r_{X1} and r_{X2} for each $X \in \{A,B,C,D\}$. In other words, the network gets incomplete information about property values.

Fig. 2.16 shows property values the network outputs for each input pattern p_{Xi} after learning. In the case of conventional learning, the network can not output correct values for unknown properties. On the other hand, in the case of OLL, the network outputs correctly for unknown properties except for r_{C1}. This means that the network find classes of input patterns correctly by the converging effect of OLL, so that the network increases the generalization ability.

2.4 Summary

In this chapter, a new learning method, *overload learning*, is proposed. In this method, an additional random task is merged into a given original task in order to adjust the network size and the complexity of tasks. This method has the following effects:

- It reduces active dimensions of hidden patterns for the original task.
- It converges the cluster of patterns.

These effects provide the following merits:

- We can get the minimum number of dimensions of hidden patterns for a given task. It will be useful to determine the suitable size of networks for tasks. Moreover, a network acquires suitable representation by reducing dimensions of pattern representation. As a result the generalization ability increases.

	target output		w rand. task		w/o rand. task	
input	q_{Xi}	r_{Xi}	q_{Xi}	r_{Xi}	q_{Xi}	r_{Xi}
p_{A1}	...*	...*	...*	...*	...*	-*..
p_{A2}	...*	...*	...*	...*	...*	=+..
p_{A3}	...*	...*	...*	...*	...*	...*
p_{A4}	...*	...*	...*	...*	.*..	...*
p_{B1}	..*.	..*.	..*.	-.*.	..*.	-.+.
p_{B2}	..*.	..*.	..*.	..*.	..*.	-.+.
p_{B3}	..*.	..*.	..*.	..*.	..*.	..*.
p_{B4}	..*.	..*.	..*.	..*.	.+*.	..*.
p_{C1}	.*..	.*..	.*..	-...	.*..	-..*
p_{C2}	.*..	.*..	.*..	.*..	.*..	...+
p_{C3}	.*..	.*..	.*..	.*..	.*..	.*..
p_{C4}	.*..	.*..	.*..	.*..	.=..	.*..
p_{D1}	*...	*...	*...	*...	*...	-...
p_{D2}	*...	*...	*...	*...	*...	=...
p_{D3}	*...	*...	*...	*...	*...	*...
p_{D4}	*...	*...	*...	*...	.=-.	*...

. .000~ .125
- ~ .375
= ~ .625
+ ~ .875
* ~1.000

Figure 2.16: Output Values for Each Pattern.
Patterns with underlines are not given to the network as teacher signals during learning.

- Converged clusters are useful to analyze patterns of a hidden layer as symbols. They also cause generalization of tasks.

There also remain the following open problems:

- A priority of tasks in learning is controlled by parameter α in a penalty function (2.1). The value of α is set empirically. We need to develop a technique to set this parameter automatically.
- This learning method should be applied to other models. This learning method is very simple, so that it may be easy to apply to various network models.

Chapter 3

Pattern Representation of Sequence

3.1 Introduction

Information which humans process is complex. In symbol processing, in the field of artificial intelligence, such complex information is often arranged in data structures like *lists* in LISP. The information is represented and processed flexibly through operations of such structures. On the other hand, patterns processed by neural networks can not represent such structures explicitly. As a result it is difficult for neural networks to process such complex information directly.

The variety of the size of data that represent such information is one of the major problems that arise from trying to process structured information by neural networks. While the size of structured data used in symbol processing varies, neural networks generally process fixed-sized patterns. Temporal-sequence processing is a way to process variable-sized data by a processor that operates fixed-sized data: Variable-sized data are divided into fixed-sized fragments. Then the processor operates on these fragments one by one in a certain order.

Many researchers have been trying to apply layered neural networks with recurrent links to process temporal sequences. Elman analyzed behavior of such networks when they learned a prediction task of various sequences[Elm88]. Cleeremans et.al. tried to train networks to learn finite state grammars[CSSL89]. In these works, networks were trained by the back-propagation method to minimize output errors within one time step. Such a method is simple enough so that it is biologically plausible. However, it has a disadvantage that it is difficult to learn complex temporal-sequence processing, especially with long distance dependencies (LDD). One way to avoid the disadvantage is to use

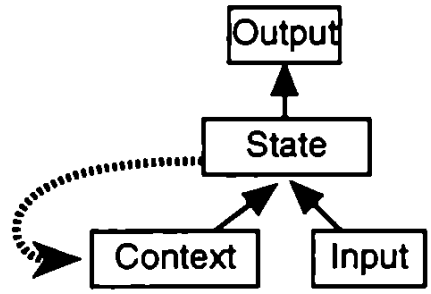


Figure 3.1: Simple Recurrent Network.

the back-propagation through time (BPTT) method [WZ89]. This method, however, has another disadvantage that it requires to record whole states during processing. Such a mechanism is not plausible biologically.

In order to process temporal-sequences with LDD, a network needs a mechanism to hold information about inputs through time. In this chapter, I focus information about input histories that is represented by activity patterns of neural networks, and propose two methods to solve the above disadvantage.

3.2 Simple Recurrent Networks

3.2.1 Elman's Networks

Elman proposed a model of simple recurrent networks (SRN) shown in Fig. 3.1 for learning sequential tasks. This model consists of *input*, *context*, *hidden* and *output* layers and links between input and hidden layers, context and hidden layers, and hidden and output layers. And also, it has a feedback connection from the hidden layer to the context layer, which copies patterns on the hidden layer to the context layer for the next time step.

This network works like a three-layered feed-forward network within one time step: Initially an input pattern is set into the input layer and a hidden pattern at the previous time step is set on the context layer by the feedback connection. Activations of units on hidden and output layers are calculated in the following manner.

$$input_i(t) = \sum_{j \in U_i} w_{ij} o_j(t) + \theta_i \tag{3.1}$$

$$o_i(t) = \phi(input_i) \tag{3.2}$$

$$\phi(x) = \frac{1}{1+\exp(-x/T)} \tag{3.3}$$

where t indicates a time step; i, j are identities of units; $input_i(t)$, $o_i(t)$ and θ_i are input, activation and threshold values of unit i respectively; U_i is a set of units connect to unit i; w_{ij} is a weight of a link from unit j to unit i; and T is the temperature parameter of the network.

In a learning phase, a desired output pattern is given to the output layer as a target in each time step. The network changes weights of links and threshold values of units by the back-propagation method to minimize the following penalty function:

$$E = \sum_{i \in \text{output layer}} (o_i(t) - \bar{o}_i(t))^2 \tag{3.4}$$

where $\bar{o}_i(t)$ is a desired output value of unit i. In this learning, error information is back-propagated within one time step and *not* back-propagated through time by using feedback connections from the hidden layer to the context layer.

Elman reported that SRNs found structures in time of sequential tasks through learning [Elm88]. For example, when an SRN is trained to learn *the prediction task* [1] of XOR sequences [2], the network found that the length of a fundamental cycle of this task was 3, and in one cycle, the first and the second data were random value and the last data was the exclusive-OR of these two values. Cleeremans et. al. applied similar networks to a task of predicting sequences that generated by an regular grammar. They showed the ability that SRNs found the same state-transitions as of a finite state automaton that corresponds to the regular grammar[CSSL89].

3.2.2 Disadvantage of Simple Recurrent Networks

SRNs, however, have a disadvantage that they can not find '*long distance dependencies* (LDD)' in sequential tasks. An LDD is a phenomena that an input effect outputs long time after the input occurs in a sequential task. An embedded structure of a complex

[1] A task to predict next data of given sequences in each time step. In the case when a given sequence is $a_1, a_2, a_3 \ldots$, an SRN outputs a_{t+1} when it receives a_t in time step t.

[2] A sequence of binary data that is constructed by randomly concatenating examples of input/output pairs of XOR, that is 2-bit input and 1-bit output. A sample of an XOR sequence might be:

1 0 1 0 0 0 0 1 1 1 1 0 ...

sentence in a natural language is an example of LDDs. In a complex sentence, there are some correlations like *subject-verb agreement* between both sides of an embedded clause as follows:

"*The dog* which chased cats *is* mine",
"*Dogs* which chased cats *are* mine".

Conventional SRNs do not have the ability to learn sequential tasks with LDDs, because they are trained by back-propagation to minimize output errors within one time step. For example, consider an experiment in which an SRN is trained to achieve a prediction task of an 'n-sequence' shown in Fig. 3.2 (experiment-1). An n-sequence is cyclic. In one cycle the first symbol is P_0, followed by one of $\{a, b, c, d, e\}$ (called a 'pre-embedded symbol'), a sequence '$P_1 P_2 \cdots P_n$' (called an 'embedded sequence'), the same symbol as the pre-embedded symbol (called a 'post-embedded symbol'), and a sequence '$P_{n+1} \cdots P_m$'. In other words, a cycle of an n-sequence has an LDD that there is a correlation between a pre-embedded symbol and a post-embedded symbol over an embedded sequence. In this experiment, each symbol in n-sequences was represented by a pattern shown in Fig. 3.3. The SRN consists of 15 input units, 50 context units, 50 state units and 15 output units. Initial weights of links were set randomly in the range of $[-1, 1]$. Learning was done independently for each $n = 0 \sim 9$. After learning, I recorded outputs of the network at the timing when it predicted a post-embedded symbol of each cycle, and calculated accuracies [3] of the prediction. Fig. 3.4 shows how the accuracy changes when n increases. As shown in this graph, the accuracy of the prediction goes down suddenly.

In order to process temporal sequences that have LDDs like n-sequences, an SRN must retain information about histories of inputs in patterns of a context layer. In the case of n-sequences, information about a pre-embedded symbol must be retain in patterns of a context layer until a network predicts a post-embedded symbol. In learning of the experiment-1, however, it is not considered how to retain such information. Therefore the information is lost, so that the network can not predict post-embedded symbols correctly when embedded sequences are long.

In the following sections, I discuss how to measure information about input histories retained in patterns on the context layer and propose two methods to retain such information effectively.

[3] Correlation coefficients of outputs and targets.

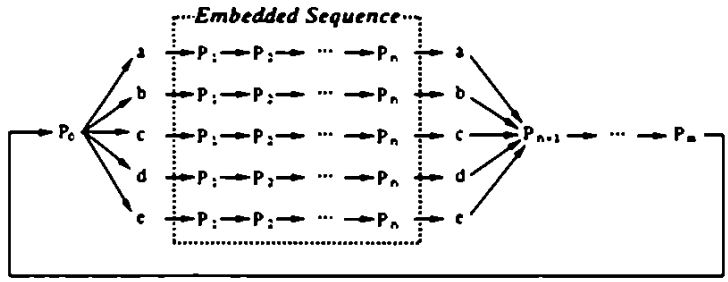


Figure 3.2: n-sequence

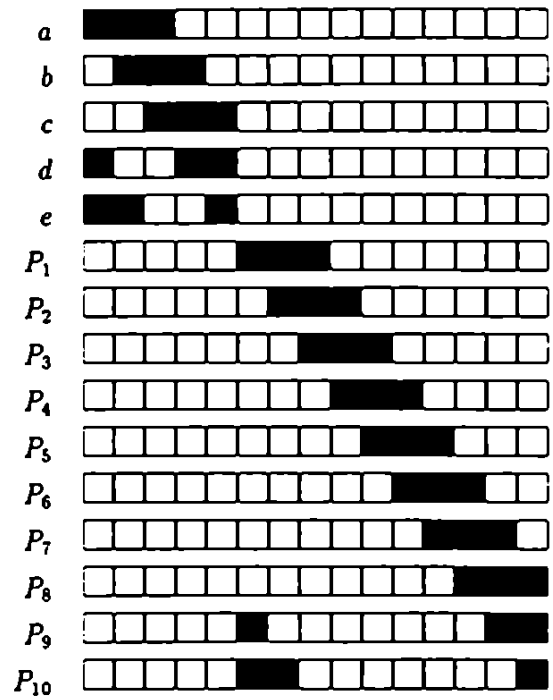


Figure 3.3: Patterns of Symbols in n-sequence

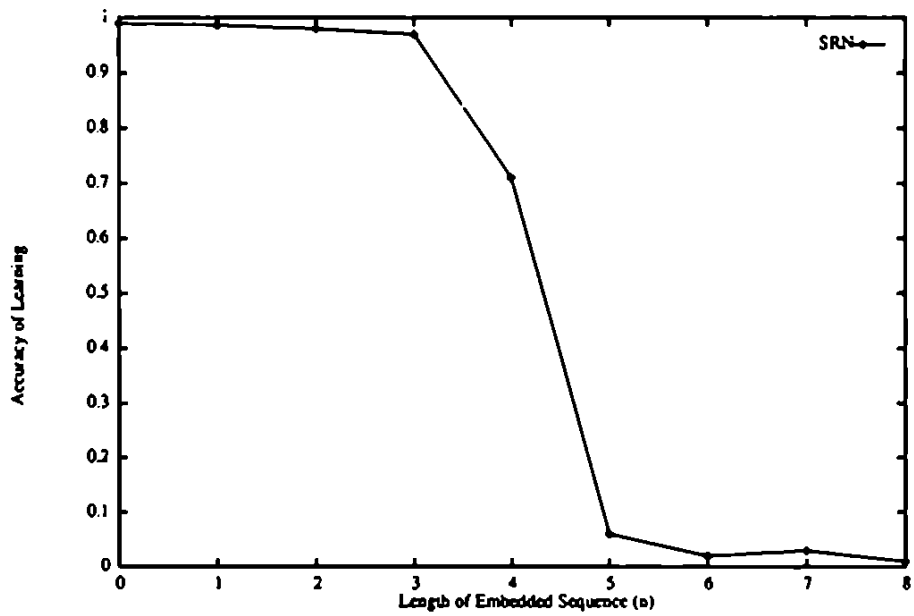


Figure 3.4: Performance of Learning n-sequence by SRNs

3.3 Retaining Information in Difference between Patterns

3.3.1 Information and Difference of Patterns

All information is represented by activation patterns in a neural network. However, it is meaningless that all patterns occur in the network are the same, because it outputs only constant patterns for any inputs. For meaningful processing, the pattern must vary. In other words, information is retained in the difference between patterns. Hence I focus the distance between patterns as a measure of difference of patterns, that is a measure how much information is represented by patterns.

To start out, let's examine a change of the distance between corresponding patterns of a context layer while an SRN is processing two kinds of cycles of a '7-sequence' problem. Initially, I recorded a sequence of context patterns while the network was processing an a-cycle [4] after an a-cycle. This sequence was treated as a base sequence. Then, I recorded another sequence of context patterns during a b-cycle after an a-cycle ($a \rightarrow$b). This sequence was compared with the base sequence in order to calculate distances of corresponding context patterns of these sequences. I also recorded sequences of context patterns in the case of an a-cycle after a b-cycle ($b \rightarrow a$) and in the case of an a-cycle after an a-cycle ($a \rightarrow a$, this is different from the base sequence), and calculated distances between corresponding patterns of each of them and the base sequence. Fig. 3.5 shows changes of these distances in a cycle. In this graph, $time = 2$ is a timing when the network receives pre-embedded symbols, and $time = 9$ is a timing when the network predicts post-embedded symbols. At a timing of $time = 2$, the distance between $a \rightarrow b$ and the base sequence is large enough, while the distance between a pattern of $a \rightarrow a$ and the base sequence is small. However, the distance between $a \rightarrow b$ and the base sequence becomes smaller and smaller during the processing of embedded sequences. Finally, at a timing of $time = 9$, the distance between $a \rightarrow b$ and the base sequence is comparable to the distance between $a \rightarrow a$ and the base sequence. This means that a pattern in each cycle becomes almost the same with each other after processing embedded sequences. Therefore the network can not predict post-embedded symbols correctly.

This phenomenon is caused by the fact that the distance of patterns on the hidden layer is smaller than that of patterns on the context layer when patterns on the input

[4] an 'x'-cycle means the cycle whose pre-embedded symbol is 'x.'

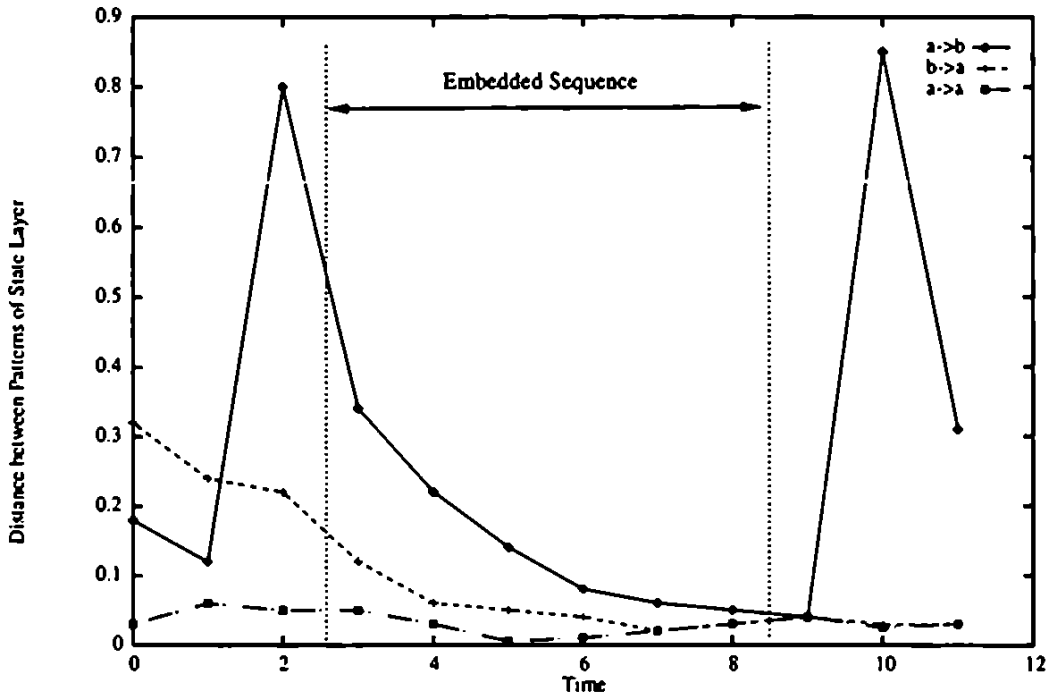


Figure 3.5: Change of Distance between Patterns of State Layer during Embedded Sequences

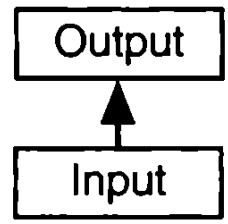


Figure 3.6: Pattern Transformation Network

layer are the same. In the following section, I discuss how to avoid this phenomenon.

3.3.2 Transformation of Patterns and Change of Distance between Patterns

Consider a pattern-transformation-network like Fig. 3.6 which consists of input and output layers. Assume that the following conditions about the pattern-transformation-network hold.

- The distribution of w, a weight of a link between input and output layers, is a Gaussian distribution $G(w; 0, \sigma_w)$, whose mean is 0 and variance σ_w^2.
- The average activity, β, of input patterns is a constant.

$$\beta = \frac{|(\text{input pattern vector})|^2}{(\text{number of dimensions of input pattern})} = \text{constant} \tag{3.5}$$

- All threshold values of units in the output layer are all 0.
- The output function of units in the output layer is defined as follows: [5]

$$f(x) = \int_{-\infty}^{x} G(\xi; 0, \tau) d\xi \tag{3.6}$$

Moreover, 'a *normalized distance*' between patterns x and y is defined as follows:

$$\alpha = \frac{|x - y|^2}{2 \times (\text{number of dimensions of patterns})}. \tag{3.7}$$

[5]This function is not exactly the same as (3.3), but is very similar and is assumed to be able to approximate (3.3).

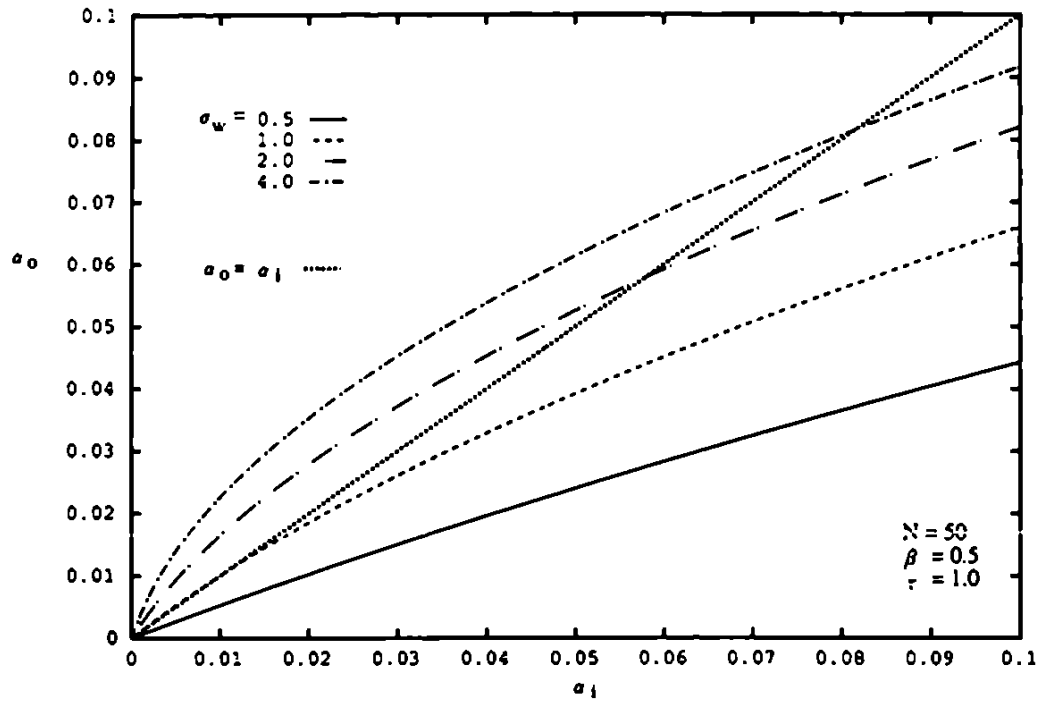


Figure 3.7: Relation between α_o and α_i

Given these assumptions, we can derive the following relation between α_i, a normalized distance of two input patterns, and α_o, normalized distance of output patterns of the network that receives the input patterns. (See appendix A for the detail.)

$$\alpha_o = \frac{1}{2\pi}\left[\cos^{-1}\left(\frac{(\beta-\alpha_i)N\sigma_w^2}{\beta N\sigma_w^2+\tau^2}\right)-\cos^{-1}\left(\frac{\beta N\sigma_w^2}{\beta N\sigma_w^2+\tau^2}\right)\right] \tag{3.8}$$

where N is the number of dimensions of input patterns. This result is a generalized version of the relation of normalized distances of input and output patterns by the transformations of random networks ([Ama78]). [6] Fig. 3.7 shows the relation between α_i and α_o for various σ_w^2 where $N = 50$, $\beta = 0.5$, $\tau = 1.0$.

Note that a differential coefficient $\partial\alpha_o/\partial\alpha_i$ at $\alpha_i = 0$ is a finite value in the case of $\tau > 0$. $\partial\alpha_o/\partial\alpha_i$ decreases as $N\sigma_w^2$ decreases. This means that if $N\sigma_w^2$ is small enough, $\partial\alpha_o/\partial\alpha_i$ at $\alpha_i = 0$ is less than 1, so that the normalized distance of patterns decreases by the pattern transformation. In this case, the normalized distance falls down to 0 by

[6] The result of [Ama78] is the case of $\tau = 0$ in (3.8).

recursive transformation of patterns by a pattern-transformation-network. As a result, all patterns become the same. On the other hand, in the case when $N\sigma_w^2$ is large enough and $\partial\alpha_o/\partial\alpha_i > 1$ at $\alpha_i = 0$, the normalized distance is attracted to a positive value by recursive transformations of patterns. This means that the difference of patterns is kept through recursive transformations.

3.3.3 Distance Keeping Method

As shown in Fig. 3.5, differences of patterns on the context layer become similar during the processing of embedded sequences of n-sequence in the experiment-1. This is the case when $N\sigma_w^2$ is small enough in (3.8). From the result of discussions in the previous section, we can find that this 'decreasing distance' problem is avoided by making $N\sigma_w^2$ large enough. In this case, a difference of patterns that occurs by receiving pre-embedded symbols is kept during processing embedded sequence. As a result, the network can predict post-embedded symbols correctly. I call this method the '*distance keeping* (DK)' method.

There are two ways to make $N\sigma_w^2$ large: to make N large or to make σ_w large. I took the latter in experiments in section 3.5. In order to do this, I set initial values of weights of links from a context layer to a state layer according to a distribution with a large variance.

3.4 Minimization of Information-Loss

In section 3.3, information represented by patterns is measured by distances between patterns. On the other hand, we can also define a measure of such information in the manner of Shannon's information theory. In this section, another method based on such a measure is described.

3.4.1 Measure Information-Loss

Consider a pattern-transformation-network shown in Fig. 3.6 again. Let $\mathcal{X} = \{x_i\}$ be a set of input patterns to the network and $\mathcal{Y} = \{y_i\}$ be a set of output patterns of the network. I also use '$\mathcal{X}$' and '$\mathcal{Y}$' as symbols to indicate information sources that provide input and output patterns respectively. In Shannon's information theory, a mutual information I

between $\mathcal{X}$ and $\mathcal{Y}$ is defined as follows:

$$\mathrm{I}(\mathcal{X},\mathcal{Y}) = \mathrm{H}(\mathcal{X}) - \mathrm{H}(\mathcal{X}|\mathcal{Y}) \tag{3.9}$$

where

$$\mathrm{H}(\mathcal{X}) = -\int_{\mathcal{X}}[\log p(\boldsymbol{x})]p(\boldsymbol{x})d\boldsymbol{x} \tag{3.10}$$

$$\mathrm{H}(\mathcal{X}|\mathcal{Y}) = -\int_{\mathcal{X}}[\log p(\boldsymbol{x}|\boldsymbol{y})]p(\boldsymbol{x})d\boldsymbol{x} \tag{3.11}$$

and $p(\boldsymbol{x})$ is a probability density, and $p(\boldsymbol{x}|\boldsymbol{y})$ is a conditional probability density of input patterns when an output pattern is $\boldsymbol{y}$ respectively. In (3.9), $\mathrm{I}(\mathcal{X},\mathcal{Y})$ means how much information about input patterns is retained in output patterns. Moreover, $\mathrm{H}(\mathcal{X})$ means a original quantity of the information. Therefore in the case when $\mathrm{H}(\mathcal{X})$ is constant, we can treat $\mathrm{H}(\mathcal{X}|\mathcal{Y})$ as a measure of loss of information about input patterns.

3.4.2 Minimum Square Error

In order to process sequence with LDD, we need a technique to minimize $\mathrm{H}(\mathcal{X}|\mathcal{Y})$. However, direct minimization of $\mathrm{H}(\mathcal{X}|\mathcal{Y})$ is difficult.

Let's consider the loss of information from another point of view. The loss of information through a pattern transformation corresponds to the degree of ambiguity when input patterns are reconstructed from output patterns. Thus, we can measure loss of information by a minimum square error of estimation of input patterns from output patterns as follows:

$$E_R = \min_{A}\int_{\mathcal{X}}\int_{\mathcal{Y}}(\boldsymbol{x} - \boldsymbol{F}(\boldsymbol{y};A))^2 p(\boldsymbol{x},\boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y} \tag{3.12}$$

where $\boldsymbol{F}$ is a vector function with parameters A.

It is easy to implement a mechanism to minimize E_R to network learning as follows. Consider a network like Fig. 3.8. This network is trained to output the same patterns on the *reconst* (= reconstruct) layer as of the *input* layer. In this case, the network solves the following optimization problem by the back-propagation method.

$$\min_{W_{IO},W_{OR}}(\boldsymbol{z} - \boldsymbol{x})^2 \tag{3.13}$$

where $\boldsymbol{z}$ is an output pattern on the reconst layer, and W_{IO} and W_{OR} are weight matrices from the input layer to the output layer and from the output layer to the reconst layer respectively. (3.13) is reformed as follows:

$$\min_{W_1}\left[\min_{W_2}(\boldsymbol{x} - \boldsymbol{z})^2\right] \tag{3.14}$$

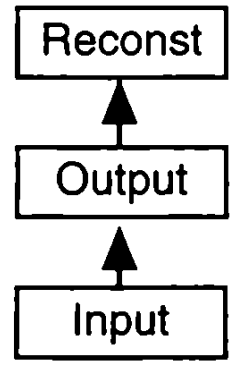


Figure 3.8: Pattern Transformation Network with Reconst Layer

In this equation, the term in $[\cdot]$ corresponds to the minimum square error E_R in (3.12). Thus learning of the network in Fig. 3.8 minimizes E_R.

3.4.3 Relation between IL and MSE

In this section, the relation between the measure of the loss of information $\mathrm{H}(\mathcal{X}|\mathcal{Y})$ and the minimum square error E_R is discussed. Suppose the following assumptions:

- A distribution of output patterns, $p(y)$, remains constant even if weights of links change by learning.
- For any output patterns y, a conditional distribution $p(x|y)$ is a multi dimensional Gaussian distribution whose covariance matrix is diagonal.

Given these assumptions, the following relation between $\mathrm{H}(\mathcal{X}|\mathcal{Y})$ and E_R holds:

$$\frac{2\mathrm{H}(\mathcal{X}|\mathcal{Y})}{N} \leq \log \frac{E_R}{N} + \log 2\pi \tag{3.15}$$

(See appendix B for more detail.) In other words, E_R is an upper limit of H. Therefore H is minimized indirectly by minimizing E_R. When the network in Fig. 3.8 is trained to minimize E_R, the loss of information by transforming patterns from the input layer to the output layer becomes small. As a result, patterns of the output layer become to represent information about patterns of the input layer effectively.

3.4.4 X Model

From the discussion in the previous section, we can derive a new learning method, called the '*information-loss minimization* (ILM)' method for SRNs. Consider a network like Fig. 3.9. This network is called '*X-model*.' In this network, *input*, *context*, *state* and *output* layers are the same as in an SRN in Fig. 3.1. *Reconst-input* and *reconst-context* layers correspond to a *reconst* layer in Fig. 3.8, a *state* layer corresponds to an *output* layer in Fig. 3.8, and *context* and *input* layers correspond to an *input* layer in Fig. 3.8.

In the learning phase, an input pattern is set into the input layer and a previous pattern of the state layer is set into the context layer. Simultaneously, as target patterns, a desired output pattern is given to the output layer and patterns of the input and context layers are given to the reconst-input and reconst-context layers respectively. Then the network is trained to minimize the following penalty function:

$$E = < (x_O - x_T)^2 + (x_{RI} - x_I)^2 + (x_{RC} - x_C)^2 > \quad (3.16)$$

where x_O, x_{RI}, x_{RC}, x_I, x_C are activation pattern vectors of the output, reconst-input, reconst-context, input and context layers respectively ; x_T is a required output pattern vector given from an external teacher.

As discussed in the previous section, the network learns to represent information about patterns of the input and context layers effectively by a pattern of the state layer. Because the pattern of the state layer becomes a next pattern of the context layer, a next pattern of the state layer also represents the information about the current patterns of the input and context layers. In this way, information about the input and context layers is represented by patterns of the state layer (also the context layer) recursively. As a result, information about input patterns is retained in patterns of the state layer for a long time. Using such information, the network becomes to be able to learn LDDs.

3.5 Experiments and Discussions

In order to show the proposed methods in the previous sections perform well, the following experiments using the same task as in experiment-1 were carried out.

3.5.1 Learning Prediction Task of n-sequences by DK Method

In order to make a variance of a distribution of weights σ_w^2 large, Weights of links from the context layer to the state layer are set according to a uniform distribution in $[-5, 5]$

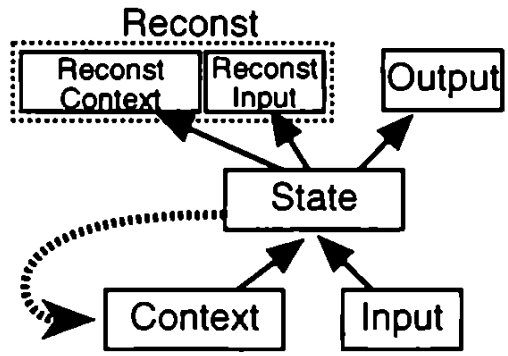


Figure 3.9: X Model

instead of $\{-1, 1\}$. Other conditions were the same as in experiment-1. After learning, I analyzed the performance of the network in the same way as in experiment-1.

The result is shown by plus marks (+) in Fig. 3.10. In this graph, while the performance in the case of the experiment-1 goes down quickly when n is greater than 4, the performance of the DK method is kept high where n is less than 7. This means that the difference of patterns of the context layer is kept during embedded sequence, so that the network can distinguish pre-embedded symbols when the network predicts post-embedded symbols.

However, the performance goes down slowly where $n > 7$. The reason is as follows: As shown in Fig. 3.7, in the case of large variances, a distance of patterns increases if an original distance is small, while a distance decreases if an original distance is large enough. This means that small differences by noise and large differences by inputs become similar and become indistinguishable from each other after recursive translations. This is unavoidable because the setting of the weights does not reflect the nature of given tasks in this method.

3.5.2 Learning Prediction Task of n-sequence by ILM Method

In the experiment of the ILM method, the number of units in the reconst layer (that is, reconst-input and reconst-context layer) is 65 and other conditions are the same as in experiment-1. The result is shown by square marks (□) in Fig. 3.10. We can find the accuracy of ILM method remains high even if n-sequences have long embedded sequences like $n = 7, 8$.

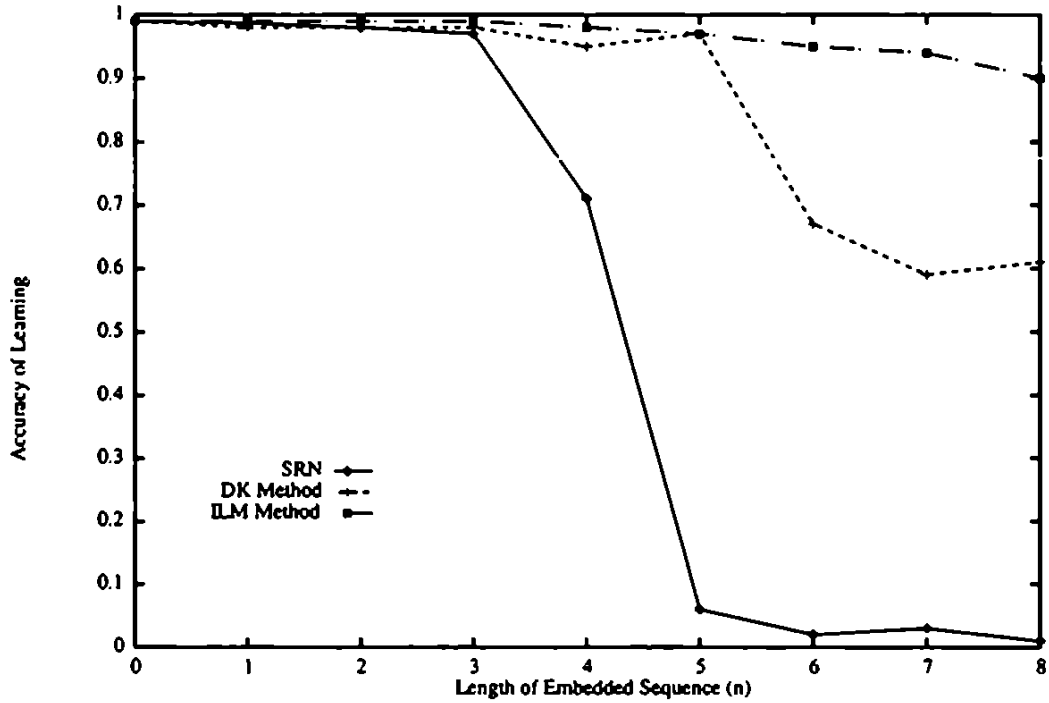


Figure 3.10: Performance of Learning n-sequence by DK Method and by ILM Method

As shown in Fig. 3.10, the performance of the ILM method is better than that of the DK method. The reason is that while weights of links are set randomly regardless of a given task in the DK method, they are tuned to a given task through learning in the ILM method. Therefore patterns on the context layer represent information about input histories more effectively in the ILM method than the DK method. However, the ILM method also has a limit. Since a network has a fixed-sized state layer, the capacity to retain information in patterns is limited. Because of the limited capacity, only a finite length of input histories are retained in the network. Therefore the network can not learn tasks that have longer distance dependencies than the finite length.

3.5.3 Comparison of Two Method

These two methods have advantages over each other.

As mentioned in the previous section, the ILM method is better than the DK method from the viewpoint of the ability in learning.

On the other hand, from the viewpoint of the speed of learning, the DK method is better than the ILM method. The reason is that while a mechanism to keep information about input histories is constructed through learning in the ILM method, it is constructed before learning in the DK method. As a result, the ILM method takes more time to learn.

From these advantages, we can consider preferred types of tasks for these methods. The ILM method is suitable for tasks with complex LDDs in which the distribution of inputs is constant. On the other hand, the DK method is suitable for tasks in which the distribution changes so that a network must adapt itself to the changes quickly.

3.6 Summary

In this chapter, I proposed two methods, the distance keeping (DK) method and the information loss minimization (ILM) method, for simple recurrent networks. These methods solve the disadvantage that it is difficult for simple recurrent networks to learn temporal sequence processing with long distance dependencies (LDD).

In the DK method, initial weights of links are set according to a distribution whose variance is large enough. By means of the large variance of weights, the difference of patterns of context layers is kept through time. As a result, a network can process sequences with LDDs.

In the ILM method, new layers are added to a given network. The network learns to output the same patterns on the new layers as those of context and input layers. This learning minimizes the quantity of loss of information about input histories indirectly, so that the network becomes to be able to process sequences with LDDs through learning.

These methods do not depend on the back-propagation method. Thus it is easy to apply them to other learning methods and network models. For example, the ILM method can be implemented to Boltzmann machines in the same manner described in this chapter[Nod89].

There remain the following open problems for these methods:

- These methods make it difficult to analyze patterns of a state layer and extract structures of information of sequential processing. These methods are derived by focusing only efficiency of representing information of input histories by patterns of the state layer. Thus, information of even useless inputs is also represented by the patterns.

- There is a certain limit to the length of LDD which networks can learn to deal with. While a longer distance dependency requires more capacity to keep input histories, the size of a hidden layer of a network is fixed. Therefore a network trained by these method can not deal with LDDs that requires more capacity than one of the network.

These problems will be solved partially in the next chapter.

Chapter 4

Learning State Transition of Finite State Transducers

4.1 Introduction

One of characteristics of simple recurrent networks is the correspondence with finite state transducers. In processing by finite state transducers, structures of information are represented by state-transitions of the transducers. Many researchers have focused on this point. They tried to analyze pattern transitions of simple recurrent networks as state-transitions of finite state transducers in order to extract structures of processing that the networks learned. In these works, there remains an open problem that networks do not entirely acquire suitable state-transitions. This problem comes from a lack of correspondence between learning of simple recurrent networks and finite state transducers.

In this chapter, a new method for simple recurrent networks to learn suitable state-transitions is proposed. The method has correspondence with the learning of finite state transducers. In order to derive it, a procedure to construct a finite state transducer from input-output examples is composed using the state-minimization technique (in section 4.2). Then each step of the method is reconstructed as a learning of neural networks (in section 4.3).

4.2 SRN and FST

4.2.1 Simple Recurrent Networks (SRN)

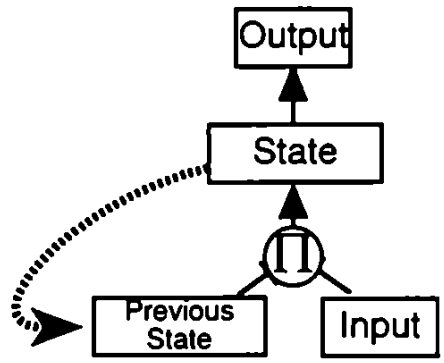


Figure 4.1: Simple Recurrent Network with Sigma-Pi Link.

Consider a simple recurrent network (SRN) like Fig. 4.1. An SRN consists of *input*, *previous-state*, *state* and *output* layers, Sigma-Pi-type links [GSC+90] [1] from input and previous-state layers to the state layer, and Sigma-type links from the state layer to the output layer. Moreover, the network has recurrent links to copy patterns of state layer to the previous-state layer with 1 time delay.

This network works in discrete time. In each time step, an external input pattern is set into the input layer and a previous pattern of the state layer is set into the previous-state layer. Then patterns of state and output layers are calculated in the manner of standard feed-forward networks.

Note that it is not necessary to use Sigma-Pi-type links among input, previous-state and state layers. It is only for avoiding a limitation of transitions of SRN [GSC+90]. [2] We can have the same discussion as in the rest of this chapter in the case of using Sigma-type links instead of Sigma-Pi-type links.

There are two strategies to train an SRN to learn a given sequential task. One strategy is to use the 'back-propagation through time (BPTT)' method to minimize output error. BPTT is powerful. However, it has a demerit that it requires to record whole status of a network during processing. Such a mechanism is not plausible biologically. Another

[1] The input value to the unit k in the state layer is calculated as follows:

$$\text{input}_k = \sum_{i \in PS} \sum_{j \in I} w_{ijk} x_i x_j$$

where 'PS', 'I' are respectively sets of units of previous-state and input layers; x_i is an output of the unit i; w_{ijk} is a weight of the link from the unit i and the unit j to the unit k.

[2] This limitation will be discussed in section 5.1.2.

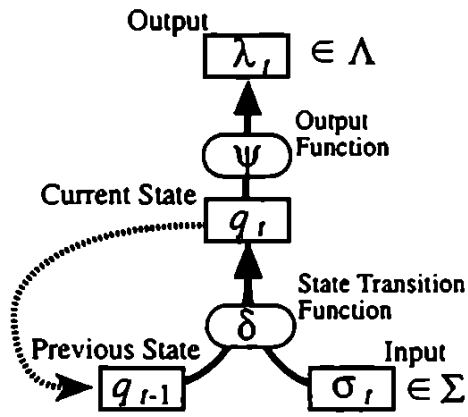


Figure 4.2: Finite State Transducer.

strategy is to use the back-propagation simply within one time step (called a simple BP, or SBP). A merit of this strategy is that learning is simple and plausible biologically. However, it has a demerit that it has poor ability to learn complex sequential tasks.

In these strategies, I take the later for learning an SRN.

4.2.2 Finite State Transducer (FST)

In the automata theory, a finite state transducer (FST) is defined by the following 6-tuple.

$$\text{FST} = < Q, \Sigma, \Lambda, \delta, \psi, q_0 >$$

where Q is a set of states, Σ is a set of input symbols, δ is a state-transition function, q_0 is an initial state, Λ is a set of output symbols, and ψ is an output function. Fig. 4.2 shows a schema of state-transitions of an FST.

4.2.3 Learning FST from given examples

When a set of examples of input-output sequences is given, we can construct an FST that performs these input-output sequences with minimum states. This is based on the state-minimization technique of FSTs[HU79]. The procedure is as follows :

[Learning FST]

S1 Make states each of which corresponds to a possible input history.

S2 Assign an output value to each state according to examples. Then group states into groups according to outputs of states.

S3 Group states in a group into sub-groups according to the group of next states after transitions from the states. Repeat this sub-grouping until no more groups are generated.

S4 Unify states belonging to the same group together a state, and reform state-transitions and outputs of each state.

(For more details, see appendix C.)

4.2.4 Correspondence between SRN and FST

As shown in Fig. 4.1 and Fig. 4.2, it is easy to consider a correspondence between an SRN and an FST. Activation patterns of input, output, previous-state and state layers respectively correspond to input symbols, output symbols, current states and next states after transitions. Links from input and previous-state layers to a state layer correspond to a state-transition function, and links from a state layer to an output layer correspond to an output function.

On the other hand, learning of SRNs described in section 4.2.1 does not correspond to learning of FSTs. Therefore, it is not sure that SRNs acquire suitable state-transitions as FSTs. As a result of acquiring unsuitable state transitions, the ability of SRNs decrease and also it becomes difficult to analyze patterns of the state layer.

In the next section, in order to solve this problem, I propose a new network model and its learning method, which corresponds to the learning of FSTs.

4.3 SGH Model

4.3.1 Network Architecture

Fig. 4.3 shows an overview of the network architecture of the proposed model, called the 'SGH model'. It consists of 'SRN', 'grouping' and 'history' modules. In learning phase, the history module is trained first, the grouping module second, and the SRN

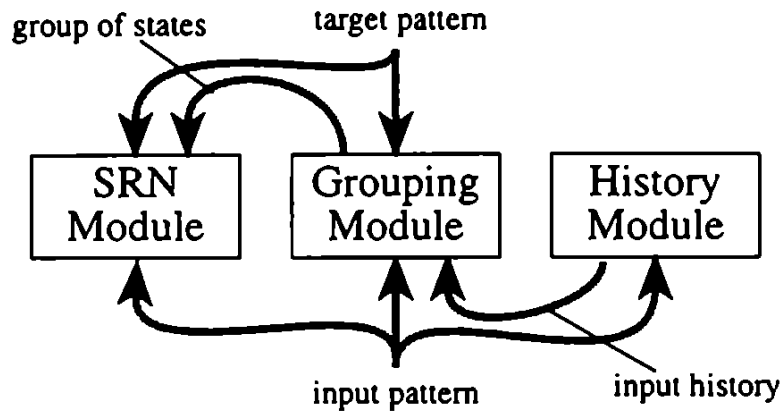


Figure 4.3: Whole Network Architecture.

module third. After learning, the SRN module works alone as an SRN. In other words, grouping and history modules are used only for learning.

The history module learns pattern representations in which information about input histories is represented by patterns effectively. This learning corresponds to the step S1 of the FST learning procedure. After learning, this module outputs patterns representing input histories to the grouping module as input patterns.

The grouping module classifies patterns of input histories into groups according to required outputs and next states. This classification is performed through learning. This learning corresponds to the step S2 and the step S3. After learning, this module outputs patterns representing groups of states to the SRN module as teacher patterns.

The SRN module learns final state-transitions and an output function according to patterns of state-groups from the grouping module and required output patterns from external teachers. This learning corresponds to the step S4. After learning, a part of this module works alone as an SRN.

In the following sections, the detail of learning of each module is described. Note that only the SBP method is used in learning of each module.

4.3.2 History Module

The learning of the history module corresponds to the step S1 in the FST learning, that is, making states corresponding to histories of input data. In order to realize this, An *X model* network with the ILM method described in chapter 3 is used. By this model

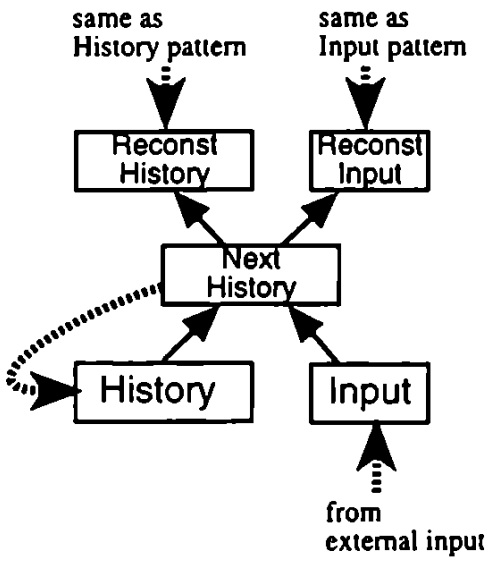


Figure 4.4: History Module.

and method, input histories can be represented by patterns. These patterns can be seen as states corresponding to input histories. Actually, I use a network like Fig. 4.4. In this network, an external input pattern is set into the *input* layer, and a pattern of the *next-history* layer in the previous time step is copied into the *history* layer. For target signals, the same patterns of *input* and *history* layers are given to *reconst-input* and *reconst-history* layers respectively.

This network trained to minimize the following penalty:

$$E_{\text{history}} = < \left| \boldsymbol{x}^t_{\text{RI}} - \boldsymbol{x}^t_{\text{I}} \right|^2 + \left| \boldsymbol{x}^t_{\text{RH}} - \boldsymbol{x}^t_{\text{H}} \right|^2 > \tag{4.1}$$

where $\boldsymbol{x}^t_{\text{RI}}$, $\boldsymbol{x}^t_{\text{I}}$, $\boldsymbol{x}^t_{\text{RH}}$ and $\boldsymbol{x}^t_{\text{H}}$ are respectively pattern vectors of reconst-input, input, reconst-history and history layers at time step t. Through this learning, information about input histories becomes to be represented by patterns of the history layer. After learning, this module outputs these patterns of the history layer to the grouping module as patterns of states correspond to input histories.

4.3.3 Grouping Module

The learning of the grouping module corresponds to the step S2 and the step S3, that is grouping states according to the output and the next state after transitions. In order to do this, I consider a technique of grouping input patterns on a hidden layer.

Table 4.1: Relation Between Difference of Input/Target Data and of Hidden Patterns.

		target data	
		same	dif.
input	same	same	dif.
data	dif.	?	dif.

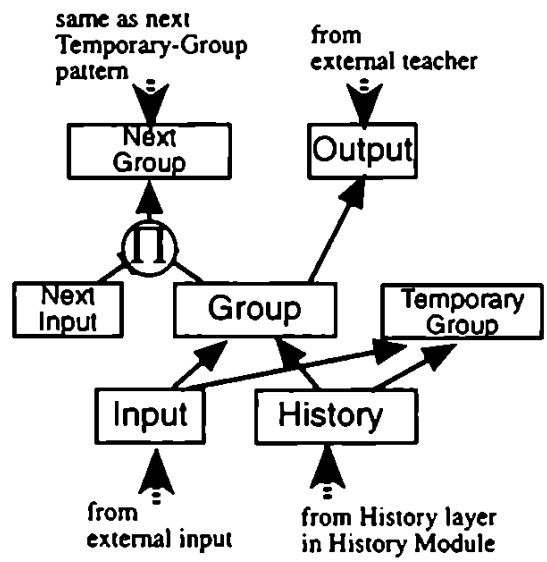


Figure 4.5: Grouping Module.

Grouping Input Patterns on A Hidden Layer

Suppose that a feed-forward three-layer network is trained by using two pairs of inputs and target outputs. We can not know what patterns will occur on the hidden layer after learning. However, we can know whether two hidden patterns for the two inputs are the same or not. Tab. 4.1 shows whether hidden patterns for each of two inputs are the same or different in each case when two inputs are the same or different and two targets are the same or different. From this table, we can find that hidden patterns are different when inputs and targets are different. This means that inputs are grouped into patterns of the hidden layer according to the target output.

Consider a network shown in Fig. 4.5 as a grouping module. In this network, a pattern of the *history* layer in the history module is set on the *history* layer, a current external input pattern is set on the *input* layer, and a next external input pattern is set on the *next-input* layer. As target signals, a required output pattern is given to the *output* layer, and a pattern of the *temporal-group* layer at the next time step is given to the *next-group* layer. Moreover, links from *group* and *next-input* layers to the *next-group* layer are Sigma-Pi-type links. The network is trained to minimize the following penalty:

$$E_{\text{grouping}} = < \left| x_{\text{O}}^{t} - x_{\text{T}}^{t} \right|^2 + \left| x_{\text{NG}}^{t} - x_{\text{TG}}^{t+1} \right|^2 > \tag{4.2}$$

where x_{O}^{t}, x_{NG}^{t} and x_{TG}^{t+1} are respectively pattern vectors of output, next-group layers at time step t and of the temporal layer at time step $t+1$; x_{T}^{t} is a required output pattern vectors at time step t.

Note that weights of links to the temporal-group layer are copied from links to the group layer at long enough intervals compared with a time scale of weight learning (in experiments in the next section, each 5000 ~10000 epochs). Therefore patterns of the temporary-group layer are almost the same as those of the group layer, but more stable than them. This layer is used for providing stable teacher patterns for the next-group layer.

As mentioned above, patterns of history and input layers are grouped into patterns of the group layer according to x_{T}^{t} and x_{TG}^{t+1}. This means that states that correspond to input histories are grouped into groups represented by patterns of the group layer according to output of the states. This corresponds to the step S2. Also, states are grouped according to the group of next states after transitions from the states, because links to the temporal-group layer are copies of links to the group layer, so that x_{TG}^{t+1} indicates a group of a next

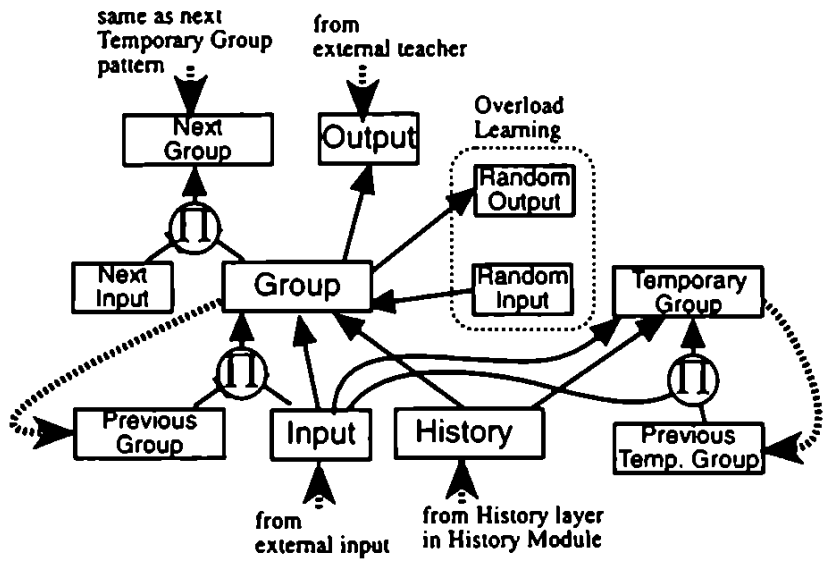


Figure 4.6: Actual Grouping Module.

state. This corresponds to the step S3. [3]

After learning, this module outputs patterns of the group layer to the SRN module as patterns of groups of states.

Inhibition of Redundant Groups

As shown in Tab. 4.1, it is not assured that patterns of the hidden layer become the same in the case of different inputs and the same target outputs. Therefore some redundant grouping of states may occur. In order to avoid this redundant grouping, the OLL method described in chapter 2 is used. The OLL method inhibits redundant grouping because the OLL method has an effect to eliminate redundant pattern representation of hidden layers. This effect is enhanced by installing a new intermediate layer between input and hidden layers. Moreover, it is empirically known that grouping of states becomes effective if the grouping layer receives previous patterns of the group layer. For these reasons, a network shown in Fig. 4.6 is actually used for the grouping module. (For simplicity intermediate layers are eliminated in this figure.)

[3]Conceptually, it works in the same manner in the case of using patterns of the group layer instead of the temporal-group layer as a target of the next-group layer. However, learning progresses gradually, and the network can not get suitable target signals about groups of next states in the middle of learning. Use of the temporal-group layer avoids this and provides stable target signals.

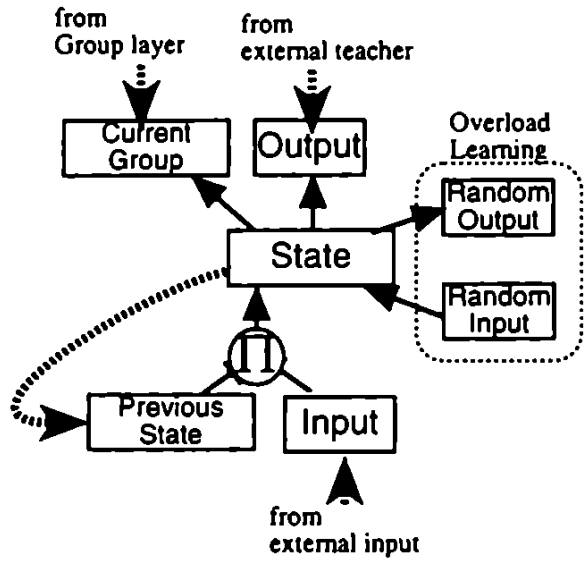


Figure 4.7: SRN Module.

4.3.4 SRN Module

The SRN module constructs reduced state transitions by using information about groups of states from the grouping module. In order to do this, a network like Fig. 4.7 is considered. This network is the same as a SRN in Fig. 4.1 except for the *current-group* layer. In this network, an external input pattern is set on the *input* layer and a pattern of the state layer at the previous time step is set on the *previous-state* layer. As target signals, a required output pattern is given to the *output* layer and a pattern of the *group* layer in the grouping module is given to the *current-group* layer. The network is trained to minimize the following penalty:

$$E_{\mathrm{SRN}} = < \left| x_{\mathrm{O}}^t - x_{\mathrm{T}}^t \right|^2 + \left| x_{\mathrm{CG}}^t - x_{\mathrm{G}}^t \right|^2 > \tag{4.3}$$

where x_{CG}^t is a pattern vector of the current-group layer at time step t. Moreover, in order to avoid generating redundant states, the OLL method is used on the state layer. By the effect of the second term of the right side of (4.3), each pattern of the state layer comes to have a one-to-one correspondence to a pattern of the group layer in the grouping module. As a result, each state of the SRN part corresponds to a group of states that have the same output and the same next states. In other words, states in a group unified into one state. This corresponds to the step S4. The module also constructs an output function by minimizing the first term.

4.4 Experiments

4.4.1 Learning Process of Grouping Module

In order to demonstrate how a grouping module groups states correspond to input histories, the following experiment (Ex.1) was carried out.

Consider a finite state automaton that has state-transitions shown in Fig. 4.8. This automaton generates four sequences, that is 'AEFGHI', 'BEFGHI', 'CEFGHJ' and 'DEFGHJ'. An SGH network learns a *sequence prediction task* [4] using these sequences. If the network acquires the same state-transitions as Fig. 4.8, we can say that the learning is successful. In this experiment, each of history, next-history and reconst-history layers consists of 30 units, each of group, previous-group, temporal-group, next-group, state, previous-state and current-group layers consists of 6 units, and each of input and output layers consists of 10 units. In input and output layers, each symbol in Fig. 4.8 was represented as localist representation [5].

I recorded patterns of the group layer during learning and analyzed them by principle component analysis (PCA). Fig. 4.9 shows the process of learning of the grouping module. Each graph shows changes of the first principle component of patterns of the group layer during the processing of each of sequences in a certain stage of learning. In these graph, each sequential position corresponds to states in Fig. 4.8 as follows: The sequential position 0 corresponds to state q_0; 1 corresponds to q_1 and q_2; 2 corresponds to q_3 and q_4; 3 corresponds to q_5 and q_6; 4 corresponds to q_7 and q_8; 5 corresponds to q_9 and q_{10}. Fig. 4.9-(a) is a stage when the network has grouped states according to outputs. In this stage, state q_9 and state q_{10} in Fig. 4.8 are represented by different patterns. But each pair of $\{q_1, q_2\}$, $\{q_3, q_4\}$, $\{q_5, q_6\}$ and $\{q_7, q_8\}$ is represented by the same pattern because two states of each pair have the same output. It means two states of each pair are grouped into the same group. In a stage of Fig. 4.9-(b), q_7 and q_8 come to be represented by different patterns because th next states of these states, that is q_9 and q_{10}, are represented by different patterns. In the same manner, states of each pair of $\{q_1, q_2\}$, $\{q_3, q_4\}$ and $\{q_5, q_6\}$ come to be represented by different patterns in stage (c), stage (d) and stage (e) respectively. Finally the network acquires patterns of groups each of which corresponds

[4] The task to predict successive elements of a sequence. When a sequence $\{x_1, x_2, x_3, x_4 \ldots\}$ is given, the network receives x_t as an input and learns to output x_{t+1} at time t.

[5] In localist representation, each unit corresponds to a symbol one-by-one, and just one unit that corresponds to a symbol to represent is activated.

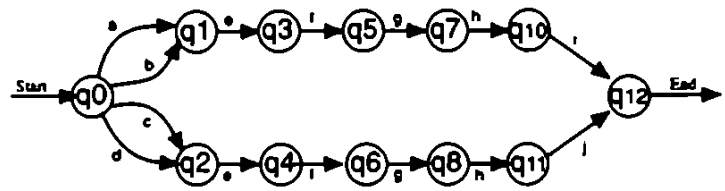


Figure 4.8: State-Transition of An Automata for Ex.1.

to a state in Fig. 4.8 one-by-one. Subsequently the SRN module started to learn and constructed an SRN that had the same state-transitions as Fig. 4.8.

4.4.2 Learning Flip-flop

The second experiment is learning the same state-transitions as of a flip-flop like Fig. 4.10. A task a network learns is that the network receives a random binary ('0' or '1') sequence followed by a terminal symbol ('s'), and then outputs a parity of the number of '1's in the sequence. In this experiment, each of history, next-history and reconst-history layers consists of 30 units, each of group, previous-group, temporal-group, next-group, state, previous-state and current-group layers consists of 10 units, each of input layers consists of 3 units and each of output layers consists of 2 units.

After learning, I analyzed patterns of the state layer by PCA. Fig. 4.11 shows an example of a result of PCA. In this figure the first and second principle components of a pattern of the state layer at each time step are plotted. As shown in this figure, the SRN module of the network acquired the same state-transitions as Fig. 4.10.

For comparison, I also trained SRNs with 60 hidden units by SBP, SRNs with 10 hidden units by BPTT, and X models with 50 hidden units. Fig. 4.12 shows the average output error of each model for various lengths of binary input sequences. We can see that errors of SRNs by SBP and X models increase suddenly when input sequences become long, while errors of SGH models and SRN by BPTT are kept small even for long input sequences. The cause of this advantage of SGH models and SRNs by BPTT is that these networks acquire the same state-transitions as Fig. 4.10. Note that SGH models use the SBP method rather than back-propagating error information through time like the BPTT method.

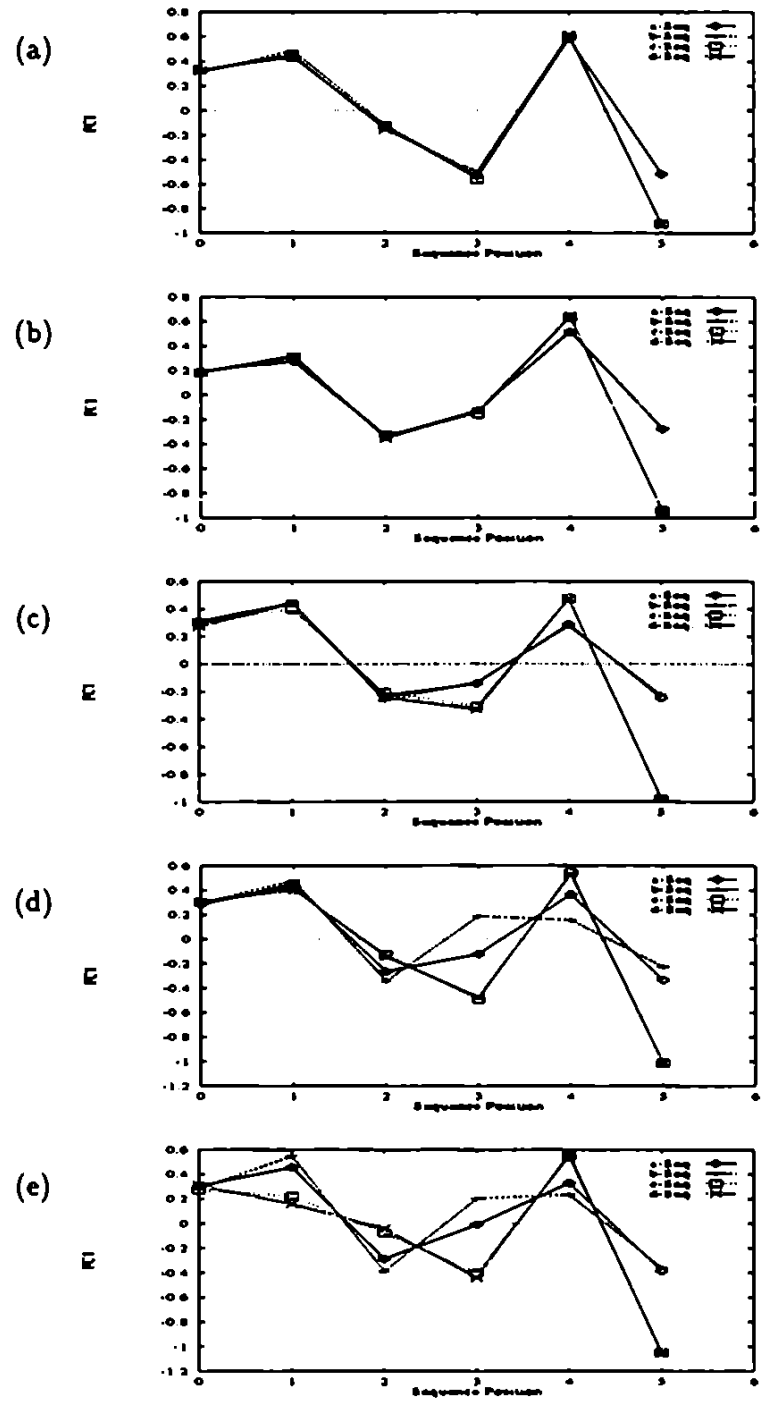


Figure 4.9: Learning Process of Group Layer in Grouping Part.

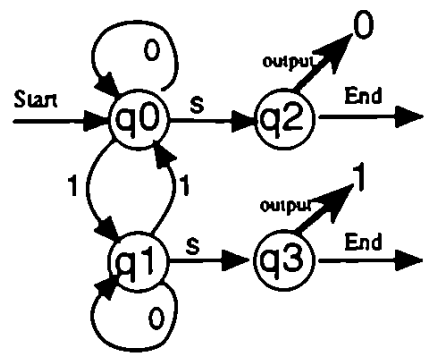


Figure 4.10: Flip Flop.

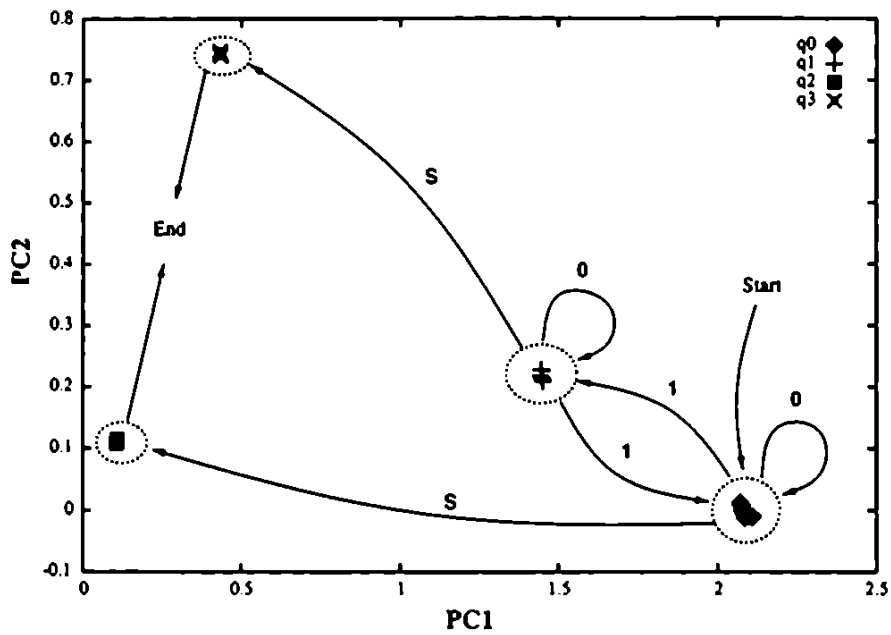


Figure 4.11: Example of State-Transition of Flip-flop.

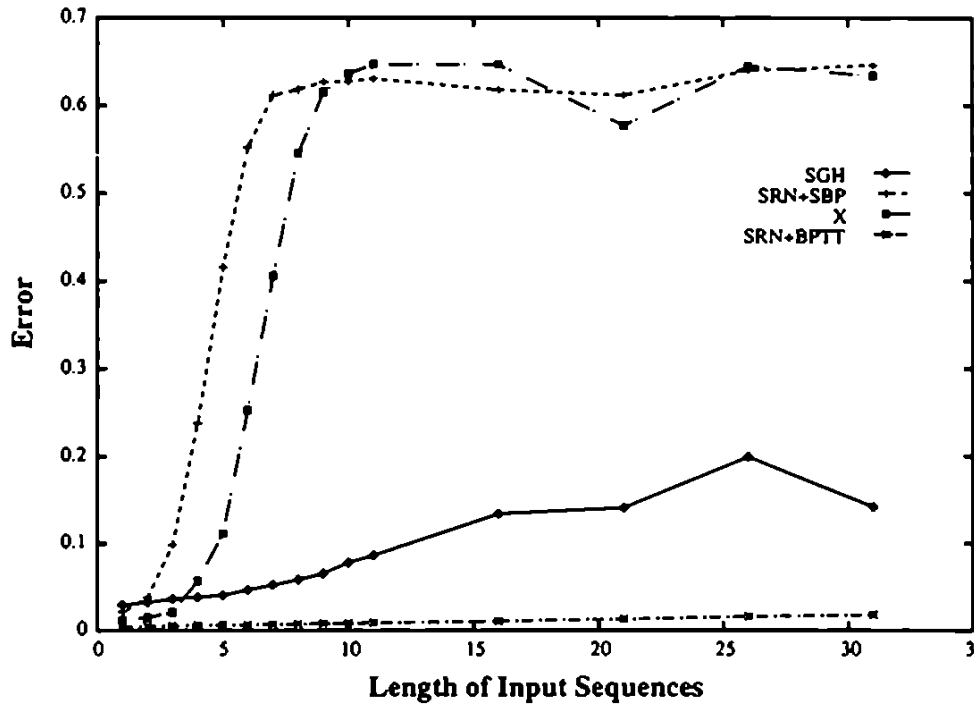


Figure 4.12: Output Error of Flip-flop.

4.4.3 Learning of Processing with Long Distance Dependency

In the field of natural language processing, one important problem is how to deal with long distance dependencies (LDD). For example, agreement between subject and main verb holds even if subordinate clauses are embedded between them like

The dog that chased cats *is* mine.
Dog that chased cats *are* mine.

I tested the ability of SGH models to deal with LDDs as follows. Consider sequences generated by an automaton that has state-transitions shown in Fig. 4.13. If the first data of the sequence is 'a' or 'b', the last data of the sequence is 'i'. Conversely, the first data is 'c' or 'd', the last data is 'j'. Between these correspondences, There are embedded sequences generated by automaton whose state-transition has loops. In other words, these sequences have LDDs. An SGH model is trained to achieve a sequence prediction task using them. In order to predict the last data of the sequences correctly, the network must retain the information about first data in it through embedded sequences. One solution of retaining such information is to acquire the same state-transitions as Fig. 4.13.

In this experiment, each of history, next-history and reconst-history layers consists of 60 units, each of group, previous-group, temporal-group, next-group, state, previous-state and current-group layers consists of 15 units and each of input and output layers consists of 9 units. I also trained an X model with 75 hidden units for comparison. Fig. 4.14 shows average output errors of both models at predicting last data. We can see that errors of X models increase for longer embedded sequences, while errors of SGH models are kept small.

Although the X model was proposed in order to deal with LDDs, the ability to deal with LDDs is limited by the capacity of the network. On the other hand, the SGH model deals with LDDs better than the X model in this case. The SGH model is aimed at acquiring suitable state-transitions. Fig. 4.15 shows an example of state-transitions which SRN modules of an SGH model acquired. This transition map is the same as Fig. 4.13. Because of these transitions, it became able to predict last data correctly even if embedded sequences become long. As this case, the SGH model has an ability to deal with very long distance dependencies that is caused by loop-type embedded stat-transitions.

In addition to it, SGH model solve another problem of the X model. Pattern transitions of the state layer in the SGH model is simple as shown in Fig. 4.15, so that it is easy to analyze such transitions in order to find structures of learned sequential tasks.

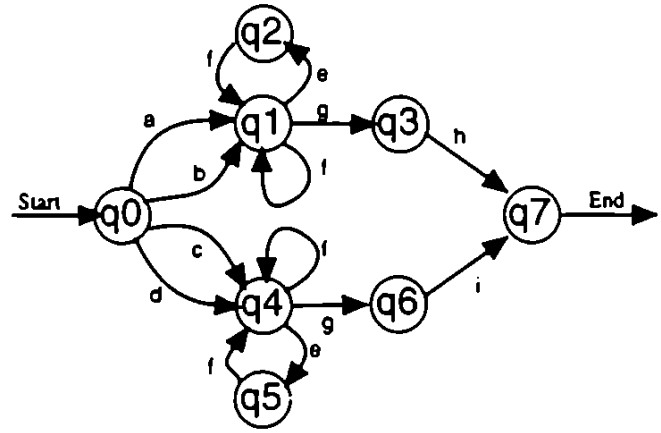


Figure 4.13: State-Transition with Embedded Loop.

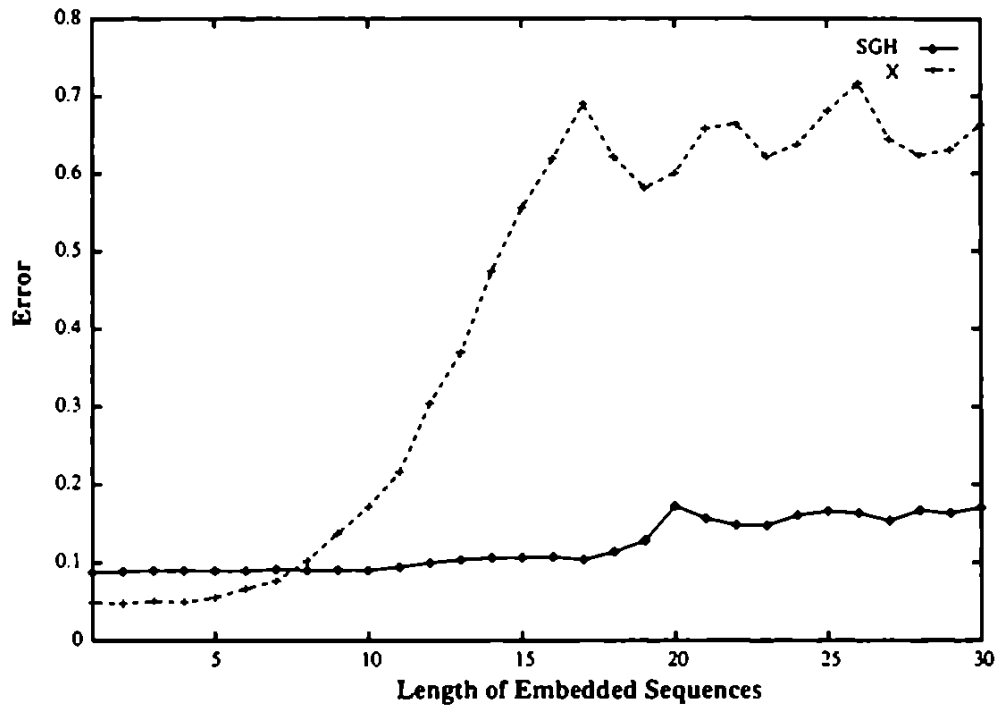


Figure 4.14: Prediction Error after Embedded Sequences.

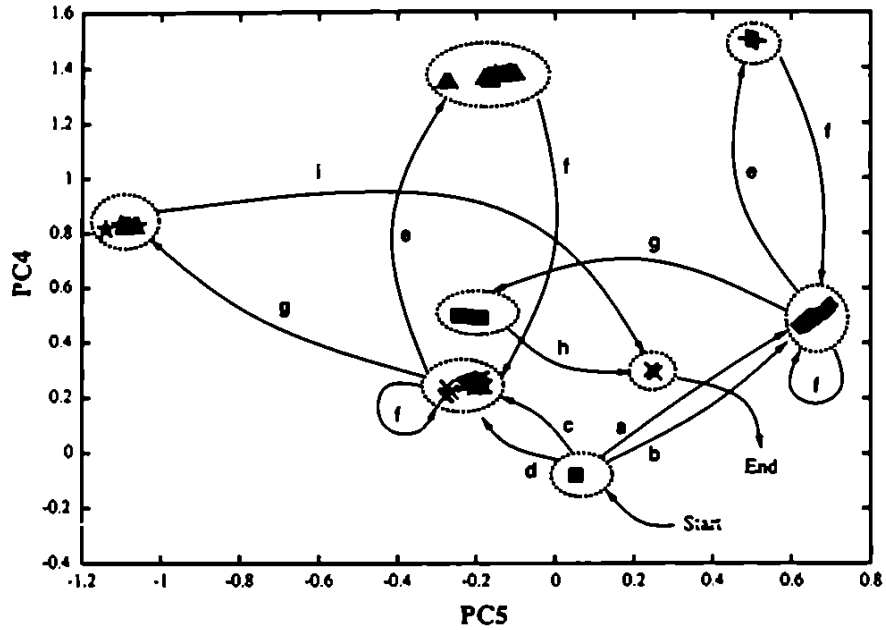


Figure 4.15: Example of State-Transition after Learning.

4.5 Summary

In this chapter, a model, called the SGH model, and its learning method to acquire suitable state-transitions are proposed. The method is derived from a procedure to construct a finite state transducer from input-output examples using the state-minimization technique. The algorithm consists of three steps: The first step is to generate states correspond to input histories. The second step is to group the states into groups according to outputs and next states of each state. The third step is to unify states in the same groups. Three network modules and learning methods for them are reconstructed from these three steps and combined into the SGH model.

The proposed method has the following features:

- The learning procedure used in this method is the simple back-propagation method, in which an error signal may not be back-propagated through time. Moreover, proposed learning methods are independent from the learning procedure, because only penalty functions to minimize are modified. Therefore it is easy to apply another learning procedure to this method.

- Penalties used in learning are calculated by signals that generated in the network itself except for required outputs. Therefore no other teachers or observers are required for learning.

These features are fit for biological models of brains and also suit for implementing into hardwares.

There also remain the following open problems:

- We must carefully choose learning parameters. Especially the timings to switch learning stages are important, because the learning of each module must be done one by one. Automatic methods to switch them should be examined.

- Pattern representation of input histories in the history module is a major factor in determining the ability of the model. An X model network generates pattern representation that represents input histories effectively. Such representation, however, is not always suitable for learning of the grouping module. We need to try to combine various types of methods to represent input histories in order to improve the ability of the model.

Chapter 5

Discussion

5.1 FST versus SRN

As mentioned in section 4.2, simple recurrent networks (SRN) have the same structure as finite state transducers (FST). However, their abilities are slightly different. In this section I discuss advantages and disadvantages of SRNs as compared to FSTs.

5.1.1 Advantage of SRN

It is well known that major advantages to neural networks are the robustness to noise and the ability to deal with analog values. In addition to it, I focus on another advantage, which is concerned with the generalization ability of learning.

Sequences used in experiments in chapter 3 and chapter 4 are able to be generated by regular grammars (RG). On the other hand, there are more powerful classes of grammars than RG. Context free grammars (CFG) are one of those classes. They are widely used in symbol processing, because it is powerful enough and also easy to process by computers. Theoretically, it is hard for SRNs to process CFGs perfectly, because processing of CFGs requires an infinite stack memory. However, we can discuss whether SRNs can deal with features of CFGs. Here, I focus on one feature of CFGs, called *sub-grammars*.

In a natural language, there are local syntactic structures that are free from context. For example, in English, a noun phrase in a sentence has a certain syntactic structure that is free from changes in the rest of the sentence. These local structures are called 'sub-grammars'. In order to represent a sub-grammar in CFG, we usually define a non-terminal symbol for the sub-grammar, and write rules that have the symbol in the left

hand. For example, noun phrases are represented as follows:

$$\begin{aligned} \mathrm{NP} &\rightarrow \mathrm{DET} \cdot \mathrm{N} \\ \mathrm{NP} &\rightarrow \mathrm{DET} \cdot \mathrm{ADJ} \cdot \mathrm{N} \\ &\vdots \end{aligned}$$

We can embed such a sub-grammar in positions where sentences should have a syntactic structure defined by the sub-grammar. For example, a subject part in a sentence has the same structure as NP and also each object part in a verb phrase has the same structure as NP. We can explain this by writing the following rules.

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{NP} \cdot \mathrm{VP} \\ \mathrm{VP} &\rightarrow \mathrm{V} \cdot \mathrm{NP} \cdot \mathrm{NP} \end{aligned}$$

On the other hand, in the case of RG, we need to define a non-terminal symbol of NP for each position as follows:

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{NP}_1 \cdot \mathrm{VP} \\ \mathrm{VP} &\rightarrow \mathrm{V} \cdot \mathrm{NP}_2 \cdot \mathrm{NP}_3 \\ \mathrm{NP}_1 &\rightarrow \mathrm{DET}_1 \cdot \mathrm{N}_1 \\ \mathrm{NP}_1 &\rightarrow \mathrm{DET}_2 \cdot \mathrm{ADJ}_1 \cdot \mathrm{N}_2 \\ \mathrm{NP}_2 &\rightarrow \mathrm{DET}_3 \cdot \mathrm{N}_3 \\ &\vdots \end{aligned}$$

Such simpleness of representing sub-grammars is one of major reasons why CFG is used in natural language processing.

The difference of representation of a sub-grammar in CFG and RG causes a difference of the generalization ability by learning. For example, if a system, which processes English sentences by CFG, learns a new syntactic structure of a subject part, such a structure will be generalized as a new syntactic structure of NP. Thus the system will become able to process sentences that have such a new structure in an object part of a verb phrase. On the other hand, in the case of RG, effects of learning a new syntactic structure will be limited in a subject part.

Using this difference of generalization abilities, I consider the following setup to determine a network learns a sub-grammar successfully or not:

1. First a network learns a prediction task of sequences in $\alpha_0 A \beta_0$ (where α_0, A and β_0 are sets of sequences). [1]

2. Next the network learns a prediction task of sequences in $\alpha_1 A' \beta_1$, where A' is a subset of A.

3. Finally the network receives sequences in $\alpha_1 A''$, where A'' is a subset of A and the intersection of A' and A'' is null set.

Consider a case when the network learned to treat A as a set of sequences generated by a sub-grammar at step 1. In this case, the network will become able to generalize to process sequences in $\alpha_1 A \beta_1$ at step 2. Therefore, the network will predict sequences in β_1 at step 3 because A'' is a subset of A. Otherwise, the network will predict sequences in β_0 or other random patterns.

In the actual experiment, a network learned the prediction task of sequences generated by a grammar shown in Fig. 5.2, where P was a sub-grammar to learn. In the experiment, the network learned grammar G (corresponds to $\alpha_0 A \beta_0$ in the previous paragraph) at first, grammar H (corresponds to $\alpha_1 A' \beta_1$) at second and finally the network received sequences generated by grammar H' (corresponds to $\alpha_1 A \beta_1$). A simplified SGH model shown in Fig. 5.1 was used as a network to learn this task. In this model, a SRN module and a temporal-group layer in a grouping module in an SGH model are removed for simplicity. Moreover a next-group layer is trained to output next patterns of a group layer instead of the temporal-group layer.

Fig. 5.3 shows responses (predictions of a next input) of the network when it receives sequences of H that the network has not learned. We can see that the network predicts correct next data in most cases. In cases of the 3rd line (predicting 'c_1') and the 4th line (predicting 'd_1') in Fig. 5.3, outputs for correct predictions are weak, but they are stronger than other outputs. From this result we can say that simplified SGH models can learn sub-grammars to a certain degree.

Why can the network learn sub-grammars? It is speculated that a topology of patterns makes it possible. In the automata theory, states of FST are represented by symbols, between which no relations are defined. Therefore it is impossible to characterize state-transitions and to define similarities between state-transitions by using relations of states. On the other hand, states of SRN are represented by patterns, between which a topology

[1] $\alpha_0 A \beta_0$ means a set of sequences that are generated by concatenating three sequences in α_0, A and β_0.

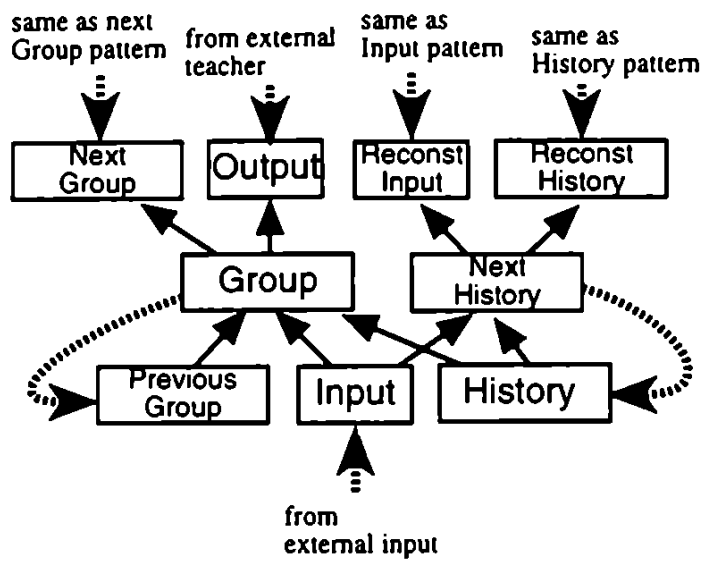


Figure 5.1: Simplified SGH model

is defined. Therefore state-transitions can be defined as trajectories in the pattern space. Thus it is easy to define similarities between these trajectories by their directions or length of. Using these similarities the network can manage state-transitions independently from actual states. As a result, it can learn sub-grammars.

5.1.2 Disadvantage of SRN

A disadvantage of SRN is that state-transitions are constrained. For example, an SRN shown in Fig. 3.1 can not realize state-transitions like Fig. 5.4, because these transitions are the same as mappings of exclusive-OR. Such mapping can not be realized by two-layer networks [MP69, GSC+90]. State-transitions of a flip-flop used in section 4.4.2 are also this type of transitions. We can escape this problem by using Sigma-Pi-type links like in chapter 4 or by putting in additional hidden layers. However, even if we use these techniques, other constraints of state-transitions arise.

This disadvantage is caused by pattern representation of states. As mentioned in section 5.1.1, pattern representation has a topology and the topology constraints transitions of states. In other words, topology of state-representation brings both advantage in the generalization ability and disadvantage in the realization ability.

$$
\begin{aligned}
G &\rightarrow g_i P g_i | g_i G g_i (i = 1, \ldots, 6) \\
H &\rightarrow a_0 P_0 a_1 | b_0 P_1 b_1 c_0 P_2 c_1 | \\
&\quad\ d_0 P_3 d_1 e_0 P_4 e_1 P_5 f_0 P_6 f_1 \\
H' &\rightarrow a_0 P a_1 | B_0 P B_1 C_0 P C_1 | \\
&\quad\ d_0 P d_1 e_0 P e_1 P f_0 P f_1 \\
P &\rightarrow Z p_0 | p_1 Z p_2 | p_2 Z p_1 | p_3 Z p_3 \\
P_0 &\rightarrow Z p_0 | p_1 Z p_2 \quad P_3 \rightarrow p_1 Z p_2 | p_2 Z p_1 \\
P_1 &\rightarrow Z p_0 | p_2 Z p_1 \quad P_4 \rightarrow p_1 Z p_2 | p_3 Z p_3 \\
P_2 &\rightarrow Z p_0 | p_3 Z p_3 \quad P_5 \rightarrow p_2 Z p_1 | p_3 Z p_3 \\
Z &\rightarrow z | z Z
\end{aligned}
$$

Figure 5.2: Sample grammar.

Input Sequence	Correct Prediction	Output of the O Layer a_1	b_1	c_1	d_1	e_1	f_1
$a_0 p_3 z z z p_3$	a_1	■					
$b_0 p_3 z p_3$	b_1		■				
$\rightarrow b_1 c_0 p_2 z p_1$	c_1			·			·
$d_0 p_3 z p_3$	d_1				·		
$\rightarrow d_1 e_0 z p_0$	e_1					■	
$\rightarrow e_1 f_0 p_1 z p_2$	f_1						■

'$\rightarrow$' indicate the sequence continued from the upper row.

Figure 5.3: Result of a local grammar learning.

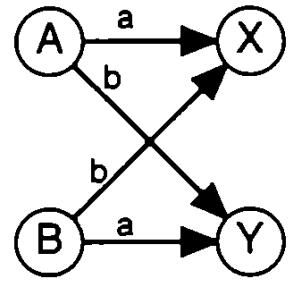


Figure 5.4: A State-Transition Which SRN Can Not Learn

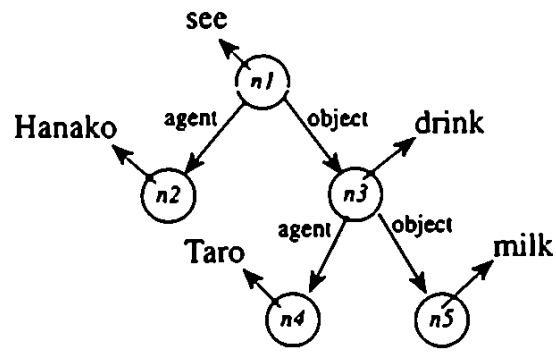


Figure 5.5: Example of Semantic Network.

5.2 Representation of Semantic Networks

This work is aimed at developing methods of neural networks to process complicated information such as information processed by symbol processing. In this section, a prototype of a method to represent semantic networks by using results of this work is described.

5.2.1 Formalization of Semantic Networks

Semantic networks are a framework to represent structured information schematically. Fig. 5.5 shows an example of semantic networks. As shown in Fig. 5.5, a semantic network consists of nodes that represent things or events [2] and links that represent relations

[2]Nodes also represent concepts in some formalization. But in the formalization described here nodes represent only individual instances of things or events.

between nodes. Formally a semantic network is defined as a 5-tuple as follows:

$$\text{Semantic Network} = < \mathcal{N}, \mathcal{T}, \mathcal{P}, r, p >$$

where $\mathcal{N}$ is a set of nodes, $\mathcal{T}$ is a set of types of relations, $\mathcal{V}$ is a set of property-values, r is a map of relations, and p is a map of properties. For example, the semantic network in Fig. 5.5 is defined a tuple $< \mathcal{N}, \mathcal{T}, \mathcal{V}, r, f >$ where

$$\begin{aligned}
\mathcal{N} &= \{n_1, n_2, n_3, n_4, n_5\} \\
\mathcal{T} &= \{\text{agent}, \text{object}\} \\
\mathcal{V} &= \{\text{Hanako}, \text{see}, \text{Taro}, \text{milk}\} \\
r &: r(n_1, \text{agent}) = n_2 \\
&\quad r(n_1, \text{object}) = n_3 \\
&\quad r(n_3, \text{agent}) = n_4 \\
&\quad r(n_3, \text{object}) = n_5 \\
p &: p(n_1) = \text{see} \\
&\quad p(n_2) = \text{Hanako} \\
&\quad p(n_3) = \text{drink} \\
&\quad p(n_4) = \text{Taro} \\
&\quad p(n_5) = \text{milk}
\end{aligned}$$

In addition to this formalization, *focuses* are added on to the semantic networks. In a semantic network, just one node is focused. The focus can move from a node to a node along a relation between them. We can access only properties of a focused node from the outside of the semantic network. The function of a semantic network with a focus is formalized as follows:

> *Initially, an initial focused node is given to a semantic network. Then as it receives types of relations one by one, it moves the focus along the type of relation, and outputs properties of a focused node.*

In other words, I treat a semantic network as a black-box that receives types and outputs properties one by one.

Using this formalization, we can consider the following correspondence between semantic networks and finite state transducers: Nodes, types and properties correspond

to states, inputs and outputs respectively. Relation and property maps correspond to state-transition and output functions respectively. In other words, a semantic network is treated as a chart of state-transitions of a finite state transducer. Under this correspondence, semantic networks can be constructed in the same way as learning of finite state transducers. Thus learning of a semantic network is formalized as follows:

> *To construct a semantic network from examples of sequences of pairs of relation-types and properties that the semantic network is required to process.*

5.2.2 Realization of Semantic Networks by Neural Networks

As mentioned in section 4.2, a simple recurrent network has the same structure as a finite state transducer. Therefore we can represent a semantic network by a simple recurrent network as shown in Fig. 5.6 according to the formalization in the previous section. In this representation, nodes, types of relations, properties are represented by patterns of node, type and property layers respectively. A relation map is represented by a link from the pre-node and type layers to the node layer. A property map is represented by a link from the node layer to the property layer.

As mentioned above, a semantic network is regarded as state-transitions of finite state transducers. Moreover, simple recurrent networks that have suitable state-transitions can be constructed by learning of SGH models. Therefore we can get a simple recurrent network that represents a semantic network by learning of an SGH model. In this learning, sequences of pairs of relation-types and properties are given to the SGH model and the SGH model acquires state-transitions whose structure is the same as a semantic network to learn.

5.2.3 Experiments

In order to show that this formalization of semantic networks performs well, a simple experiment was carried out.

Consider a semantic network like Fig. 5.7. In this network, properties of nodes 'X', 'Y', 'Z' and 'W' are selected in the following sets:

$$\begin{aligned} X &: \{\text{Human}, \text{Thing}\} \\ Y &: \{a, b, c, d\} \\ Z &: \{\text{Large}, \text{Middle}, \text{Small}\} \end{aligned}$$

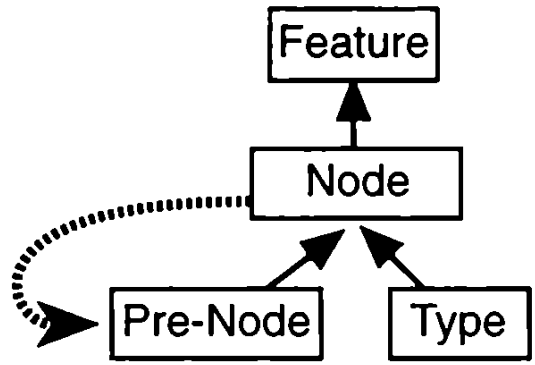


Figure 5.6: Simple Recurrent Network for representing Semantic Network.

$$W : \{Yes, No\}$$

where there is a correlation between properties of node 'X' and node 'W' as follows: When a property of 'X' is 'Human', a property of 'W' must be 'Yes'. Conversely when a property of 'X' is 'Thing', a property of 'W' must be 'No'. Thus there are 24 combinations of networks. 21 combinations of them are used for generating a training set of example sequences of type-property pairs, and other 3 combinations are used for a test set. A simplified SGH model with 20 units in the group layer and 30 units in the history layer was used to learn the semantic network. Fig. 5.8 shows output patterns of the property layer after learning. Each line in Fig. 5.8 shows a property that a neural network outputs when its state comes to each node of 'X', 'Y', 'Z' and 'W'. Fig. 5.8-(a) is a case of a combination in the training set and (b) is a case of a combination in the test set. In both cases the network outputs the property that the network memorized. This means that the network works as a semantic network as shown in Fig. 5.7. Moreover, although properties of node 'W' were not given when the network memorized a semantic network, the network outputs suitable properties of node 'W'. This means that the network found the correlation between properties of node 'X' and node 'W' through learning, and inferred a property of 'W' from a property of 'X'.

5.2.4 Discussion

The formalization of semantic networks and its realization by simple recurrent networks have the following advantages:

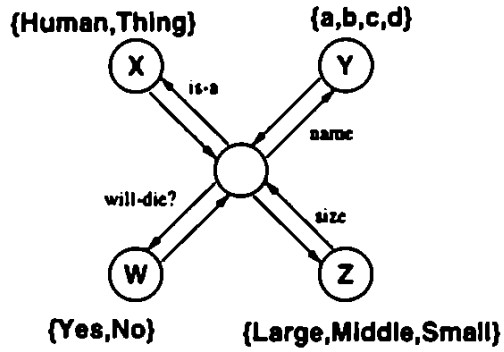
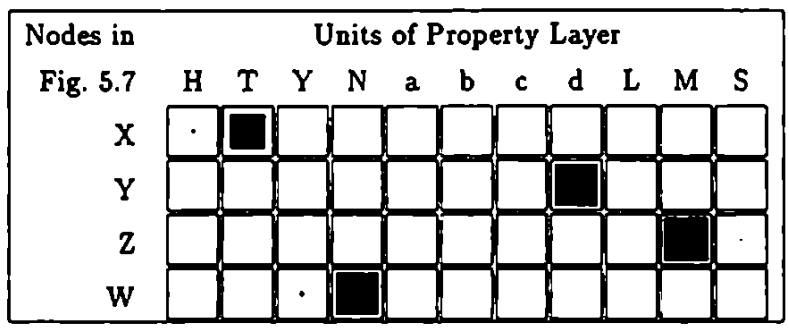
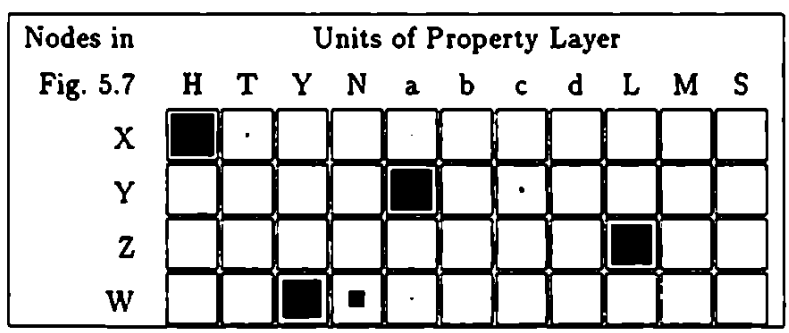


Figure 5.7: Semantic Network to Learn.



(a) X = Thing , Y = d , Z = Middle (in the training set)



(b) X = Human , Y = a , Z = Large (in the test set)

Figure 5.8: Outputs of Property Layer for Each Node

- In the conventional implementation of semantic networks to neural networks like [WP85, Sha88], neural networks can not operate or learn semantic structures, that is relations of nodes, directly. Therefore additional mechanisms to operate relations are required outside neural networks. On the other hand, in the formalization described here, a network can operate and learn semantic structure. This will provide the possibility of constructing a system to process semantic information by neural networks.

- While information in semantic networks is represented in maps of relations in symbol processing, it is represented by patterns of node layers in this formalization. As mentioned in section 5.1.1, patterns have a topology. Using the topology, similarities between information are defined naturally. Such similarities and topology provide the ability of generalization. Inference of properties of node 'W' in the experiment in section 5.2.3 is an example of this generalization ability.

  We can get a more powerful ability of generalization by supposing more strong topology of pattern representation of information. For example, the network will become able to perform multi-step inferences when pattern transitions from the pre-node layer to the node layer are liner. Too strong topology, however, decreases the ability of representation as semantic networks. Therefore we must balance the tradeoff between these abilities.

A disadvantage of this formalization is:

- Concepts can not be dealt with in this formalization. Nodes in this formalization can represent only concrete things or events. One way to deal with concepts is to represent them as properties whose patterns represent a hierarchy of concepts.

5.3 Biological Plausibility

The most important feature of artificial neural networks is that these networks are derived from models of the nervous system in brains. However many of them are not plausible as brain models. For example, the real time recurrent learning method [WZ89] requires that each unit must retain information about whole links in a network. It is difficult to suppose that an actual nervous system has such a mechanism.

In this section, I discuss biological plausibility of models proposed in this thesis from various points of view.

5.3.1 Network Structure

It is said that the number of neurons in a brain does not change. In the proposed models, additional layers are required for learning, but whole structures of these networks are fixed. Thus we need not change the number of units during learning.

It is also said that a neuron in a brain can perform relatively simple calculations. In the proposed models, each unit calculates weighted summations of activations of units connected by links and decides its activation from the summations according to a sigmoid function. These calculations are simple enough for a model of actual neurons.

5.3.2 Locality of Calculations

Each calculation of processing and learning in brains is generally supposed to be performed in a local area like a synapse. In the proposed models, all procedures of processing and learning are performed in links and units. Furthermore whole data used in these procedures are propagated only through links. It is easy to realize these procedures by calculations in local areas.

Time locality of calculations is also required. For example, it is difficult to realize the back-propagation through time method [WZ89] by calculations local in time. It requires to record all histories of activations of units. Such a mechanism is not plausible biologically. On the other hand, the X model requires only data in current time step. Also the SGH model requires only data in previous and current time step. Therefore it is possible to construct these models by calculations local in time.

5.3.3 Target Signals

A major problem of supervised learning is who provides target signals. In the proposed models, whole target signals are generated in the models except for required external outputs. For example, in the X model teacher signals for the reconst-input and reconst-context layers are patterns of the input and context layers respectively. In the SGH model, teacher signals for the current-group and next-group layers are patterns of the group layer at the same and the next time steps respectively. Therefore additional mechanisms to provide such targets are not necessary.

Moreover, in the case when the networks learn the sequence prediction task like in experiments described in section 3.5 and section 4.4, no external teachers are required.

In this case target signals for output layers are next input patterns. It is important that the networks can learn structures of given sequences through such simple tasks[Elm88].

5.3.4 Learning Procedure

The back-propagation method may be not plausible as biological models, because it requires slightly complicated procedures to propagate error information. Fortunately, in the proposed models only penalty functions are modified rather than learning procedures. Therefore it is easy to replace the back-propagation with another learning procedure. For example, we can use the reinforcement learning method to minimize the penalty functions.

5.3.5 Toward Biological Model

As discussed above, the proposed models are plausible as brain models. For example, we can draw out a brief structure of nervous systems of the simplified SGH model (Fig. 5.1) which learns a prediction task of input sequences. Fig. 5.9 shows the brief structure. In this figure, each large rectangle means a layer of neurons. Each small square means a delay-unit that propagates activations after one time delay. Each arrow means a connection between two layers. Furthermore, each circle means a unit that calculates differences of patterns of two layers. Using these differences, weights of connections are modified. For example, in the case of a reinforcement learning method, weights are reinforced by a certain mechanism when the differences are small. These mechanisms are simple enough to be plausible as biological models.

5.3.6 Disadvantages as Biological Models

There are the following disadvantages to the proposed models as biological models.

- While actual brains work in continuous time, the X model and the SGH model work in discrete time.

- In the SGH model, an operation to copy weights of links is required. Such a mechanism is not plausible as brain models.

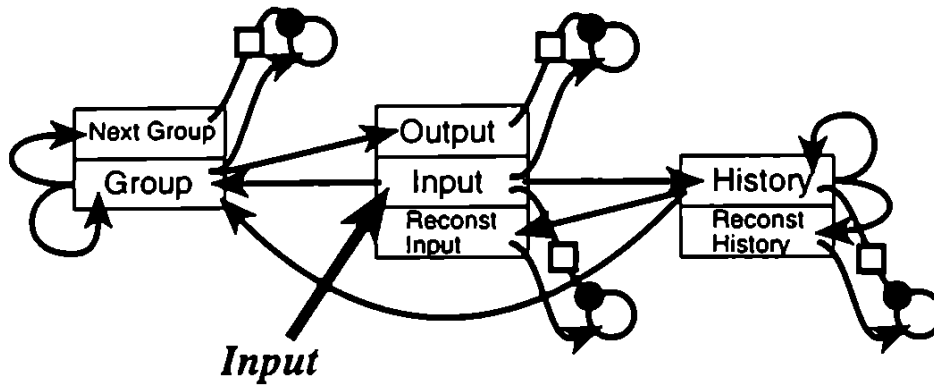


Figure 5.9: Simplified SGH Model as Brain Model

5.4 Related Works

5.4.1 Symbolization of Patterns

Many researchers try to minimize the number of hidden units by so-called a *pruning* method [Hag91, SD88, OOY93, MS89, KM91, OG93, Ree93]. A major strategy of these works is removing redundant units that are inactive or duplicate another. The pruning method, however, does not force units to become obviously redundant, so that we can not always find the minimum number of hidden units. On the other hand, the OLL method proposed in chapter 2 force units to become obviously redundant. Consequently, we can find the minimum dimensions instead of the minimum hidden units. Yet, OLL does not minimize the number of hidden units directly. We will be able to construct a more powerful pruning method by combining the OLL method.

There is another method to find simple networks, called the 'weight-decay method' [Ish89, PNH86]. In this method, complexity of networks is defined according to weights of links. The complexity is minimized through learning. As a result a reduced network is acquired. An effect of this method is different from the OLL method, because the weight-decay method minimizes mainly the number of links rather than hidden units. Thus we can use those methods complementarily.

5.4.2 Sequential Processing and Learning FST by SRN

Simple recurrent networks (SRN) are hardly studied as machines of sequential processing. Elman investigated behavior of SRNs after training a prediction task[Elm88, Elm89]. He showed that networks could find structures in sequences through a simple back-propagation learning. Similar networks are used to learn regular grammars and finite state automata [SSCM89, Pol91], and to process natural languages [MD89, JM90]. Ghahramani and Allen [GA91] proposed a model based on the same idea of an X model after an X model was proposed. Yet, they did not treat simple recurrent networks as finite state automata in learning. Thus networks can not achieve suitable state-transitions even if the networks have a capacity to realize such state-transitions. Compared with them, learning of an SGH model is based on learning of finite state transducers. Therefore an SGH model can acquire more suitable state-transitions than their models.

Giles and his colleagues are developing methods to process sequential tasks by recurrent networks and to construct minimum state-transitions of such networks [GSC+90, GCM+91a, GCM+91b, OGM92]. Main differences between my works and them are:

- They use the 'back-propagation through time (BPTT)' method to learn sequential tasks, while I use the simple BP method. The BPTT method is powerful but less plausible biologically.
- Their method to construct minimum state-transition is off-line, while my method is on-line.

5.4.3 Other Symbol Processing by Neural Networks

There are several works to represent or deal with data structures by neural networks. Pollack proposed a method to represent *CONS* cells of LISP using sand-glass-type networks[Pol89, Pol90]. Touretzky also proposed a method to represent *CONS* using Boltzmann-machines[Tou90]. Smolensky and his colleague studied the representation of data structures and variable bindings by tensor products [Smo90, LMS91]. The strategy of these works is similar to the method to represent semantic networks by neural networks described in section 5.2. In this strategy, data structures are represented as sequences of operations of neural networks. Through these operations, information is convoluted into patterns. Hinton summarized such a strategy as a 'between-level timesharing' [Hin90].

Chapter 6

Conclusions

Hybrid and integrated systems of symbol and neural processing are expected to provide flexible and robust problem solving methods. Yet it is difficult to transfer information between both processing systems because of the different type of data that used in them. From the viewpoint of neural processing, this difficulty comes from two essential characteristics of data representation in symbol processing:

- *Symbols*: each of which indicates discrete and independent information.
- *Data structures*: by which complicated information is arranged flexibly.

In this thesis I have discussed issues that arise when data are translated between symbol and neural processing modules in a hybrid system. Based on these discussions, I have proposed learning methods for neural networks to deal with symbols and data structures.

In chapter 2, a method to make pattern representation suitable for analyzing as symbols was described.

When a symbol processing module receives information from a neural processing module, pattern representation in the neural processing module must be analyzed from the view point of symbols. There are several techniques, such as clustering, for the purpose to analyze patterns as symbols. It is, however, difficult to symbolize patterns using these techniques when pattern representation is redundant. This redundant pattern representation is caused by too many hidden units for the complexity of a task.

In order to balance the number of hidden units and the complexity of the task, a new learning method, '*overload learning*', was proposed. In this method, a network is trained to learn an additional task together with the original one. Since the redundant part of

the patterns of the hidden layers is used for the additional task, minimum dimensions of hidden patterns continue to work for the original task.

Various experiments showed that the proposed method had the following effects:

- to reduce pattern representation of a hidden layer for the original task.
- to converge clusters of patterns of a hidden layer.

These effects make the symbolization of patterns easy. The concept of this method is very simple, so that it is easy to apply this method to other network structures and learning methods.

In chapter 3, learning methods for simple recurrent networks to solve the problem of how to find LDDs in temporal sequences were described.

When the neural processing module receives data from the symbol processing module, a problem of how to deal with structured data by neural networks arises. One of the major causes of this problem is the size of data. Although the size of structured data generally varies, neural networks usually process fixed-sized patterns. Temporal-sequence processing is a technique to process such variable-sized data by using processors that can process fixed-sized data. Yet, in such a technique the processors need to learn to process sequences that have long distance dependencies (LDD). In order to find LDDs, a simple recurrent network needs to retain information about input histories in patterns of a hidden layer. I formalized two measures how much information represented by the patterns.

In the first formalization, a measure of the information was defined by distances between patterns of a hidden layer. Using this measure, the '*distance-keeping*' method was proposed. In this method, the divergence of weight values of links from a context layer to a state layer is made large in order to keep distances of patterns through pattern transitions.

In the second formalization, a measure of loss of the information was defined in the manner of Shannon's information theory. Based on this measure, the '*information-loss minimization*' method was proposed. This method was derived from the relation between the measure of information-loss and mean-square-errors of an identity function realized by a three-layered network.

I carried out experiments to learn a sequential task with LDDs by using proposed methods. Results of these experiments showed that both methods increased the ability to learn tasks with LDDs by simple recurrent networks as compared with a conventional

back-propagation learning. These two methods have different features: In the 'distance-keeping' method, learning finishes quickly. On the other hand, in the 'information-loss minimization' method, networks can learn tasks with longer distance dependencies than the distance-keeping method.

In chapter 4, a method to construct suitable state-transitions of simple recurrent networks was described.

Temporal sequence processing provides another point of view for representing structured data. Simple recurrent networks have a similar structure to finite state transducers. On the other hand, in the automata theory, state-transitions of a transducer represent a structure of sequences processed by it. Simple recurrent networks, however, can not acquire suitable state-transitions by conventional learning methods. Therefore the network can not learn representation of structures of sequences.

In order to solve this problem, I proposed a network model, called the '*SGH model*', and its learning method. In order to derive the model, initially a procedure to construct a finite state transducer from examples of input-outputs was composed using the state-minimization technique. This procedure consists of three steps, the 'keeping input history' step, the 'grouping states' step, and the 'constructing state-transitions' step. Then each step was reconstructed as learning of a neural network. Finally, three networks were combined into the SGH model. By using this model, we can get a simple recurrent network that has suitable state-transitions for a given task.

I carried out some experiments to learn several kinds of state-transitions. In every case, the network acquired suitable state-transitions. Experiments also showed that it increased the ability of simple recurrent networks to process temporal sequences with LDDs.

In chapter 5, I discussed about proposed models and methods from various points of view.

First, the ability of a simple recurrent network and one of a finite state transducer were compared. Because of a topology of patterns, the flexibility of state-transitions of a simple recurrent network is limited compared with a finite state transducer. On the other hand, the topology increases the generalization ability of learning state-transitions such as *sub-grammars*. A simple experiment to learn a sub-grammar by a simplified SGH model was carried out. The result showed that simplified SGH models dealt with sub-grammars in a framework of a context free grammar.

Second, a formalization of semantic networks that were suitable for processing by the SGH model was discussed. In this formalization, a semantic network is treated as a chart of state-transitions of a finite state transducers. These state-transitions can be learned by a SGH model. Semantic networks are a framework of representation of various kinds of information used in symbol processing. Thus this formalization provides a way to combine or integrate neural and symbol processing tightly.

Finally, biological plausibility of proposed models was discussed. While artificial neural networks are originally derived from biological nervous systems in brains, many of them are not plausible as nervous systems. Proposed methods and models are simple enough and relatively plausible as biological models from various points of view. Especially, every learning methods is presented by penalty functions, each of which has a simple conceptual meaning. This makes it possible to apply the methods to various network models and learning methods, which are plausible as nervous systems.

Appendix A

Derivation of (3.8)

Let x_1, x_2 be two input pattern vectors of a pattern translator network in Fig. 3.6, and z_{1i}, z_{2i} be inputs to unit i in the output layer when the network receives input pattern x_1, x_2 respectively. Because weights of links are set randomly, z_{1i} and z_{2i} can be viewed as random variables that are independent from each other. The distribution of these numbers is a Gaussian distribution whose variance σ_{z1}^2 and covariance σ_{12}^2 are:

$$\sigma_{z1}^2 = \sigma_{z2}^2 = \sigma_z^2 = \beta N \sigma_w^2$$
$$\sigma_{12}^2 = (\beta - \alpha_i) N \sigma_w^2$$

Hence, if the number of units in the output layer is large enough, a normalized distance α_o between output patterns for x_1 and x_2, is:

$$\begin{aligned} \alpha_o &= 1/2 \int\int (f(z_1) - f(z_2))^2 p_{zz}(z_1, z_2) dz_1 dz_2 \\ &= 1/2[A(1,2) - A(1,1)] \end{aligned} \tag{A.1}$$

where

$$A(i,j) = \int\int [f(z_i)(1 - f(z_j)) + f(z_j)(1 - f(z_i))] p_{zz}(z_1, z_2) dz_1 dz_2$$

and $p_{zz}(z_1, z_2)$ is a joint probability density of z_1 and z_2. Because the output function $f(z) = \int_{-\infty}^{z} G(\xi; 0, \tau) d\xi$ can be interpreted as "a probability of the case when the sum of z and a Gaussian noise $-\xi$ is positive", $A(1,2)$ becomes :

$$A(1,2) = \int\int_{z_1' z_2' < 0} p_{zz}'(z_1', z_2') dz_1' dz_2' \tag{A.2}$$

where

$$\begin{aligned} z_1' &= z_1 + \xi_1 \\ z_2' &= z_2 + \xi_2 \end{aligned}$$

and ξ_1, ξ_2 are random variables whose distribution are independent Gaussian distribution with average = 0 and variance = τ^2, and $p'_{zz}(z'_1, z'_2)$ is a two dimensional Gaussian distribution whose center is the origin and each variance and covariance is $\sigma_z^2 + \tau^2$ and σ_{12}^2 respectively. We can calculate the definite integral in (A.2) by means of a technique used in [Ama78] as follows:

$$A(1,2) = \frac{2}{\pi} \tan^{-1} \sqrt{\frac{\alpha_i N \sigma_w^2 + \tau^2}{(2\beta - \alpha_i) N \sigma_w^2 + \tau^2}}$$

Moreover, $A(1,1)$ corresponds to $A(1,2)$ in the case of $\alpha_i = 0$. Therefore (3.8) is derived by substituting results of $A(1,1)$ and $A(1,2)$ in (A.1).

Appendix B

Derivation of (3.15)

Let $\sigma_{xi}(y)$ be a variance of a distribution of the i-th element of an input pattern vector x when an output pattern vector is y. Under two assumptions described in section 3.4.3, $\mathrm{H}(\mathcal{X}|\mathcal{Y})$ in (3.11) and E_R in (3.12) is:

$$\mathrm{H}(\mathcal{X}|\mathcal{Y}) = < \sum_{i=1}^{N} \log \sigma_{xi}(y) >_y + \log \sqrt{2\pi} \tag{B.1}$$

$$E_R = < \sum_{i=1}^{N} \sigma_{xi}^2(y) >_y \tag{B.2}$$

where $< \cdot >_y$ is the mean according to y. Hence, the arithmetic mean A_s and the geometric mean A_p of $\sigma_{xi}^2(y)$ according to i and y is:

$$A_s = < 1/N \sum_{i=1}^{N} \sigma_{xi}^2(y) >_y$$

$$A_p = \exp\left(< 1/N \sum_{i=1}^{N} \log \sigma_{xi}^2(y) >_y \right)$$

Thus (B.1) and (B.2) become:

$$\mathrm{H}(\mathcal{X}|\mathcal{Y}) = (\log 2\pi + N \log A_p)/2 \tag{B.3}$$

$$E_R = N A_s \tag{B.4}$$

On the other hand, from the Cauchy-Schwarz's inequality the following inequality holds:

$$A_p \leq A_s \tag{B.5}$$

By substituting (B.3) and (B.4) for A_p and A_s in (B.5), we can derive (3.15).

Appendix C

Procedure of Learning a FST from Examples

The procedure of learning a finite state transducer (FST) from given input-output examples consists of two sub-procedures, which are the *generation* procedure and the *reduction* procedure. The generation procedure generates an initial FST that has redundant states. The reduction procedure reduces the initial FST and constructs final FST that has the minimum number of states.

C.1 Generation Procedure

Let $\mathcal{Z}$ be a set of example sequences of input-output pairs, where

$$\mathcal{Z} = \{z^n | z^n = [z_t^n | t = 0, 1, 2, \ldots], z_t^n =< x_t^n, y_t^n > n = 0, 1, 2, \ldots, N\}$$

and x_t^n and y_t^n are respectively input and output at time t of n-th example sequences. Hence we can get an FST that realizes the same input-output responses as the given examples.

[Generation]

G1 Create an initial state q_0, label it, and assign a set $\mathcal{Z}$ and a length 0 to it.

G2 Pick a labeled state q_k and unlabel it. Let $\mathcal{Z}_k$ be an assigned set to q_k and l_k be an assigned length to q_k. Then, Classify examples z^n in $\mathcal{Z}_k$ into subsets $\mathcal{Z}_i$ according to $x_{l_k+1}^n$, that is $(l_k + 1)$-th input of example z^n. Let x_i be the $(l_k + 1)$-th input of

examples in $\mathcal{Z}_i$, that is $x^n_{l_k+1}$, and y_i be the $(l_k + 1)$-th output of examples in $\mathcal{Z}_i$, that is $y^n_{l_k+1}$. [1]

G3 Create a new set q_i for each $\mathcal{Z}_i$ created in step G2, label it, and assign a set $\mathcal{Z}_i$ and length $l_k + 1$ to q_i. Moreover, add a transition from q_k to q_i by input x_i into the state-transition map, and an output y_i of the state q_i into the output map.

G4 Repeat step G2 and G3 until no labeled states remain.

C.2 Reduction Procedure

An FST generated by the generation procedure has many redundant states. We can reduce such an FST using the '*state-minimization*' technique[HU79]. The procedure is as follows:

[Reduction]

R1 Classify all states into groups according to the output of each of the states.

R2 Pick a group, and classify states in the group into sub-groups according to the group to which the next state of transitions for each input from each of the states belongs.

R3 Repeat step R2 for all groups until no more new groups are created.

R4 Unify states belonging the same groups together into a state, and construct transitions and output functions of unified states.

[1] When the $(l_k + 1)$-th outputs of examples in a set $\mathcal{Z}_i$ are not the same, let y_i be a representative value of those outputs.

Bibliography

[Ama78] Shun-ichi Amari. *Shinkei-Kairomou no Suuri*. Sangyo Tosho, 1978.

[CSSL89] Axel Cleeremans, David Servan-Schreiber, and James L.MacClelland. Finite State Automata and Simple Recurrent Networks. *Neural Computation*, Vol. 1, pp. 372–381, 1989.

[Elm88] Jeffrey L. Elman. Finding Structure in Time. Technical Report CRL-TR-8801, Center for Research in Language, University of California, San Diego, April 1988.

[Elm89] Jeffrey L. Elman. Structured Representations and Connectionist Models. In *Eleventh Annual Conference of the Cognitive Science Society*, pp. 531–546, 1989.

[Elm91] Jefferey L. Elman. Distributed Representations, Simple Recurrent Networks, and Grammatical Structure. *Machine Learning*, Vol. 7, pp. 195–225, 1991.

[GA91] Zoubin Ghahramani and Robert B. Allen. Temporal Processing with Connectionist Networks. In *IJCNN91*, pp. II–514–546, June 1991.

[GCM+91a] C.L. Giles, D. Chen, C.B. Miller, H.H. Chen, G.Z. Sun, and Y.C. Lee. Second-Order Recurrent Neural Networks. In *IJCNN91*, pp. II–273–281, June 1991.

[GCM+91b] G. L. Giles, D. Chen, C. B. Miller, H. H. Chen, G. Z. Sun, and Y. C. Lee. Second-Order Recurrent Neural Networks for Grammartical Inference. In *IJCNN'91-Seatle*, pp. II–273–281, 1991.

[GSC+90] C. L. Giles, G. Z. Sun, H. H. Chen, Y. C. Lee, and D. Chen. Higher Order Recurrent Networks & Grammatical Inference. In *NIPS2*, pp. 380–387. Morgan Kaufmann, 1990.

[Hag91] Masafumi Hagiwara. Back-Propagation with Artificial Selection – Reduction of the Number of Learning Times and That of Hidden Units –. *Trans. of the Institute of Electronics, Information and Communication Enginieers*, Vol. J74-D-II, No. 6, pp. 812–818, 1991.

[Hin90] Geoffrey E. Hinton. Mapping Part-Whole Hierarchies into Connectionist Networks. *Artificial Intelligence*, Vol. 46, No. 1–2, pp. 47–75, 1990.

[HU79] John E. Hopcroft and Jeferey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Ish89] M. Ishikawa. A structural learning algorithm with forgetting of link weights. In *IJCNN: International Joint Conference on Neural Networks*, pp. vol.2 p.626, Jun. 1989.

[JM90] Mark F. St. John and James L. McClelland. Learning and Applying Contextual Constraints in Sentence Comprehension. *Artificial Intelligence*, Vol. 46, No. 1–2, pp. 217–257, 1990.

[KM91] John K. Kruschke and Javier R. Movellan. Benefits of Gain: Speeded Learning and Minimal Hidden Layers in back-Propagation Networks. *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 21, No. 1, pp. 273–280, 1991.

[LMS91] Geraldine Legendre, Yoshio Miyata, and Paul Smolensky. Distributed Recursive Structure Processing. In *NIPS3*, pp. 591–597. Morgan Kaufmann, 1991.

[MD89] Risto Miikkulainen and Michael G. Dyer. A Modular Network Architecture for Sequential Paraphrasing of Script-Based Stories. In *IJCNN: International Joint Conference on Neural Networks*, pp. II–49–56, 1989.

[MP69] Marvin L. Minsky and Seymore A. Papert. *Perceptrons.* MIT Press, 1969.

[MS89] Michael C. Mozer and Paul Smolensky. Skeletonization: A Technique for Trimming the Fat From A Network via Relevance Assessment. In *NIPS*, pp. 107–115. Morgan Kaufmann, 1989.

[Nod89] Itsuki Noda. Method of Learning Markov Sequence Pattern by Boltzmann Machine with Feedback Loop. Master's thesis, Faculty of Engineering, Kyoto University, 1989.

[OG93] Christian W. Omlin and C. Lee Giles. Pruning Recurrent Neural Networks for Improved Generalization Performance. Technical Report No. 93-6, Computer Science Department, Rensselaer Plytechnic Institute, Troy, N.T., April 1993.

[OGM92] C. W. Omlin, C. L. Giles, and C. B. Miller. Heuristics for the Extraction of Rules from Discrete-Time Recurrent Neural Networks. In *IJCNN'92-Baltimore*, pp. I-33-38, June 1992.

[OOY93] Takahiro Oshino, Jun Ojima, and Shinji Yamamoto. Method for Gradually Reducing a Number of Hidden Units on Back Propagation Learning Algorism. *Trans. of the Institute of Electronics, Information and Communication Enginieers*, Vol. J76-D-II. No. 7. pp. 1414-1424. 1993.

[PNH86] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Tech. Rep. CMU-CS-86-126, Carnegie Mellon Univ., 1986.

[Pol89] Jordan B. Pollack. Implications of Recursive Distributed Representations. In *NIPS1*, pp. 527-536. Morgan Kaufmann, 1989.

[Pol90] Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, Vol. 46, No. 1-2, pp. 77-105, 1990.

[Pol91] Jordan B. Pollack. The Induction of Dynamical Recognizers. *Machine Learning*, Vol. 7, pp. 227-252, 1991.

[Ree93] Russell Reed. Pruning Algorithms — A Survey. *IEEE Trans. on Neural Networks*, Vol. 4, No. 5, pp. 740-747, 1993.

[SD88] J. Sietsma and R.J.F. Dow. Neural Net Pruning - Why and How. In *Proc. of IEEE International Conference on Neural Networks*, pp. I-325-I-333, Jul. 1988.

[Sha88] Lokendra Shastri. *Semantic Networks: An Evidential Formalization and its Connectionist Realization.* Research Notes in Artificial Intelligence. Pitman, London, 1988.

[Smo90] Paul Smolensky. Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionst Systems. *Artificial Intelligence*, Vol. 46, No. 1–2, pp. 159–216, 1990.

[SSCM89] David Servan-Schreiber, Axel Cleeremans, and James L. McClelland. Learning Sequential Structure in Simple Recurrent Networks. In *NIPS1*, pp. 643–652. Morgan Kaufmann, 1989.

[Tou90] David S. Touretzky. BoltzCONS: Dynamic Symbol Structures in a Connectionist Network. *Artificial Intelligence*, Vol. 46, No. 1–2, pp. 5–46, 1990.

[WK90] Wasuhiro Wada and Mitsuo Kawato. Selection of Neural Network Structure with Generalization Capability by Using New Informatio Criterion (in Japanese). Technical Report NC90-20. The Institute of Electronics. Information and Communication Engineers, 7 1990.

[WP85] D. L. Waltz and J. B. Pollack. Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. *Cognitive Science*, Vol. 9, No. 1, pp. 51–74, 1985.

[WZ89] Ronald J. Williams and David Zipser. Gradient-Based Learning Algorithms for Recurrent Networks. In Y. Chauvin and D. E. Rumelhart, editor, *Backpropagation: Theory, Architectures and Applications*, chapter , pp. . Hillsdale, NJ: Erbaum, 1989.

List of Publications

List of Major Publications

[1] Itsuki Noda and Makoto Nagao. Learning Methods for Simple Recurrent Networks Based on Minimizing Information Loss (in Japanese). *Trans. of The Institute of Electronics, Information and Communication Engineers*, Vol. J74-D-II, No. 2, pp. 239–247, 1991.

[2] Itsuki Noda and Makoto Nagao. A Learning Method for Recurrent Networks Based on Minimization of Finite Automata. In *IJCNN'92-Baltimore*, pp. I–27–32, Jun. 1992.

[3] Itsuki Noda. Learning Method by Overload. In *IJCNN'93-Nagoya*, pp. 1357–1360, Oct. 1993.

[4] Itsuki Noda. A Learning Method for Recurrent Neural Networks Based on Minimization of States of Finite State Transducer (in Japanese). *Trans. of The Institute of Electronics, Information and Communication Engineers*,, Vol. J77-D-II, No. 11, to appear.

[5] Itsuki Noda. A Model of Recurrent Neural Networks that Learn State-Transitions of Finite State Transducers. *WCNN'94-San Diego*, pp. IV-447–452, Jun. 1994.

List of Other Publications

[1] Itsuki Noda and Makoto Nagao. Learning Method for Boltzmann Machine with Feedback Loop (in Japanese). In *Proc. of 38th Convention of IPSJ*, 3 1989.

[2] Itsuki Noda and Makoto Nagao. Learning Method of Recurrent Neural Networks base on Minimization of Information Loss (in Japanese). Technical Report NC89-

55, No. 430, The Institute of Electronics, Information and Communication Engineers, 1 1990.

[3] Itsuki Noda. Grammar Acquisition by Recurrent Neural Networks. In *Proc. Workshop on Learning '91*, 1 1991.

[4] Itsuki Noda. A Model of Recurrent Networks that Learn the Finite Automaton from Given Input-Output Sequences. In *International Symbosium on Neural Information Processing (as a part of International Symposia on Information Sciences (ISKIT '92))*, pp. 197–200, Jul. 1992.

[5] Itsuki Noda. Connectionist Symbol Processing. *Systems, Control and Information*, Vol. 36, No. 10, pp. 661–668, 1992.

[6] Itsuki Noda. Formalization of Semantic Networks for Neural Networks. In *Proc. of 3rd Convention of JNNS*, pp. 153–154, 12 1992.

[7] Itsuki Noda. Representation of Semantic Networks for Connectionist Model. In *Workshop Of Learning '92*, 1 1992.

[8] Itsuki Noda. Analogy on Connectionist Semantic Networks. In *Proc. of Workshop Of Learning '93*, 1 1993.

[9] Itsuki Noda. Cooperative Natural Language Processing. In *Proceedings of the Sixth Annural CUNY Sentence Processing Conference*, pp. 77, March 1993.

[10] Itsuki Noda. Overload Learning. Technical Report NC93–28, The Institute of Electronics, Information and Communication Engineers, 7 1993.